

Razvoj primijenjene programske potpore

10. Refleksija

Refleksija

- Proces kojim program može pregledavati i modificirati vlastitu strukturu tijekom izvođenja
- Refleksija se upotrebljava za:
 - dohvat metapodataka (sadržanih u atributima)
 - otkrivanje tipa podatka
 - pristup informacijama o učitanim asemblijima i tipovima definiranim unutar njih (pregled interakcija i instanciranje tipova)
 - dinamičko povezivanje na svojstva i postupke
 - pozivanje svojstava ili postupaka dinamički instanciranog objekta na temelju otkrivenog tipa (*dynamic invocation*), npr. *bind* fonta ili boje
 - stvaranje i korištenje novih tipova za vrijeme izvršavanja programa
- Primjeri upotrebe
 - razvoj aplikacija za reverzno inženjerstvo
 - razvoj preglednika razreda
 - razvoj editora svojstava razreda (kao prozor *Properties*)
 - dinamičko uključivanje dodataka (*pluginova*)

Važniji razredi i prostori imena za refleksiju

- Prostor imena *System.Reflection*
 - sadrži razrede i sučelja za dohvat informacija o tipovima i članovima programskog koda pri izvođenju, kao i dinamičko kreiranje tipova
 - `Assembly` – pruža informacije o asembliju i omogućava dohvat podataka o tipovima definiranim unutar njega
 - `MemberInfo` (apstraktni razred)
 - Omogućuje pristup metapodacima o članovima razreda te dinamičko pozivanje postupaka
 - `ConstructorInfo`, `PropertyInfo`, `MethodInfo`, ...
 - Pristup metapodacima konstruktora, svojstava, odnosno postupaka
- Razred *System.Type*
 - pruža informacije o nekom tipu podatka (puni naziv, je li apstraktan, javan, iz kojeg tipa je izveden, informacije o tipovima koji su korišteni za parametrizaciju, ...)
 - služi za dohvat informacija o članovima razreda (atributi, svojstva, postupci, ...)

Stvaranje objekta razreda *System.Type* (1)

- Ako je razred prisutan (referenciran) u trenutku kompilacije:
 - Pozivom postupka `GetType` na nekom postojećem objektu:

```
MojRazred r = new MojRazred();  
  
...  
Type t = r.GetType();
```
 - Korištenjem operatora `typeof`


```
Type t = typeof(MojRazred);
```
 - Korištenjem statičkog postupka `GetType` iz razreda `Type` uz navođenje punog imena naziva razreda

```
Type t = Type.GetType("MojProjekt.MojRazred");
```

Stvaranje objekta razreda *System.Type* (2)

- Ako razred nije prisutan (referenciran) u trenutku kompilacije:
 - Korištenjem statičkog postupka `GetType` iz razreda `Type` uz navođenje punog imena naziva razreda pripadajućeg asemblija (odvojen zarezom i razmakom)
 - `Type t = Type.GetType("MojProjekt.MojRazred, MojProjekt");`
 - Ako je razred generički iza naziva imena dodaje se znak ``` i broj tipova s kojima je razred parametriziran
 - `Type.GetType("System.Collection.Generic.Dictionary`2");`
 - Postupak `Type.GetType` je preopterećen te po želji omogućuje ignoriranje velikih i malih slova


Dinamičko učitavanje asemblija

- Asemblij (dll ili exe) se može dinamički učitati unutar programa
- Učitavanje vrši korištenjem postupka *LoadFromAssemblyPath* nad objektom tipa *AssemblyLoadContext* (iz prostora imena *System.Runtime.Loader*)
- Potrebno navesti punu putanju do asemblija
 - Apsolutna putanja može se dobiti iz relativne pomoću *Path.GetFullPath*
- Iz učitane asemblije mogu se dobiti informacije o svim tipovima podataka ili o nekom konkretnom
- Primjer:  `Reflection \ Reflection \ Program.cs`

```
string assemblyLocation = " ... LottoImplementation.dll";
AssemblyLoadContext loadctx = AssemblyLoadContext.Default;
Assembly asm = loadctx.LoadFromAssemblyPath(
    Path.GetFullPath(assemblyLocation));

Type type = asm.GetType("LottoImplementation.Lotto");
...
```

Naknadno povezivanje (engl. *late binding*)

- Nakon što su dostupne informacije o nekom tipu (objekt tipa `Type`) moguće stvoriti novi objekt tog tipa i pozivati njegove postupke
- Postupak *CreateInstance* unutar razreda *Activator* stvara novi objekt određenog tipa
 - Primjer:  Reflection \ Reflection \ Program.cs

```
Type type = asm.GetType("LottoImplementation.Lotto");
object obj = Activator.CreateInstance(type, 7, 39);

MethodInfo info = type.GetMethod("DrawNumbers");
object result = info.Invoke(obj, new object[] {false});

string print = string.Join(", ", (List<int>)result);
Console.WriteLine(print);
```

- Bolja varijanta: definirati sučelje koje dinamički stvoreni tip implementira (ILotto u primjeru)

Izvoz podataka u Excel (1)


- Prije još jedne demonstracije upotrebe refleksije slijedi primjer izvoza podataka u Excel



EPPlus by EPPlus Software AB

v5.5.0

A spreadsheet library for .NET framework and .NET core

- Potrebno instalirati paket EPPlus i u appsettings.json označiti da se koristi u ne-komercijalne svrhe
- Primjer:  MVC \ appsettings.json


```
"EPPlus": {  
  "ExcelPackage": {  
    "LicenseContext": "NonCommercial"  
  }  
}
```


Izvoz podataka u Excel (2)

- Primjer:  MVC \ Controllers \ ReportController.cs

```
const string ExcelContentType =  
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";  
public async Task<IActionResult> DrzaveExcel() {  
    var drzave = ... //lista s podacima  
    byte[] content;  
    using (ExcelPackage excel = new ExcelPackage()) {  
        var worksheet = excel.Workbook.Worksheets.Add("Države");  
        worksheet.Cells[1, 1].Value = "Oznaka države";  
        worksheet.Cells[1, 2].Value = "Naziv države";  
        ...  
        for (int i = 0; i < drzave.Count; i++) {  
            worksheet.Cells[i + 2, 1].Value = drzave[i].OznDrzave;  
            ...  
        }  
        content = excel.GetAsByteArray();  
    }  
    return File(content, ExcelContentType, "drzave.xlsx");  
}
```

Korištenje refleksije za izvoz u Excel (1)

- Ideja: napraviti generičko rješenje za izvoz neke kolekcije u Excel
 - Pomoću refleksije dobiti informacije o svim svojstvima nekog tipa, a zatim dinamički uzimati vrijednost pojedinog svojstva
 - Rješenje napisano kao proširenje sučelja `IEnumerable<T>`
 - Proširenje nazvano `CreateExcel` (vidi sljedeći slajd)
- Primjer korištenja:  `MVC \ ReportController.cs`

```
public async Task<IActionResult> DokumentiExcel() {  
    var dokumenti = await ctx.vw_Dokumenti  
        .  
        .ToListAsync();  
  
    byte[] content;  
    using (ExcelPackage excel = dokumenti.CreateExcel("Dokumenti")) {  
        content = excel.GetAsByteArray();  
    }  
    return File(content, ExcelContentType, "dokumenti.xlsx");  
}
```

Korištenje refleksije za izvoz u Excel (2)

- Nazivi stupaca jednaki nazivima (jednostavnih) svojstava
 - Primjer  MVC \ Extensions \ ExcelCreator.cs

```
public static ExcelPackage CreateExcel<T>(
    this IEnumerable<T> data, string worksheetName) {
    ExcelPackage excel = new ExcelPackage();
    var worksheet = excel.Workbook.Worksheets.Add(worksheetName);
    int row = 1; int col = 1;
    PropertyInfo[] props = typeof(T).GetProperties(
        BindingFlags.Instance | BindingFlags.Public);

    foreach (var prop in props) {
        if (prop.PropertyType is IEnumerable)
            continue; //preskoči kolekcije
        string name = prop.Name;
        worksheet.Cells[row, col++].Value = name;
    }
    ...
}
```

Korištenje refleksije za izvoz u Excel (3)

- Vrijednost nekog svojstva nekog objekta dobije se iz objekta tipa *PropertyInfo*

- Primjer  MVC \ Extensions \ ExcelCreator.cs


```
public static ExcelPackage CreateExcel<T>(
    this IEnumerable<T> data, string worksheetName) {
    ...
    foreach (T t in data) {
        ++row; col = 1;
        foreach (PropertyInfo prop in props) {
            if (prop.PropertyType is IEnumerable)
                continue; //preskoči kolekcije

            object value = prop.GetValue(t);
            worksheet.Cells[row, col].Value = value;
            ++col;
        }
    } ...
}
```

Atributi

- Atributi su nadodane oznake tipovima, poljima, postupcima i svojstvima
 - Definirani uglatim zagradama [] prije deklaracije entiteta koji opisuju
 - Dostupni programski tijekom izvođenja programa
- Primjeri već definiranih atributa
 - `Required` – validacijski atribut
 - `Display` – koristi se kod MVC tag-helpera *label-for*
 - `Obsolete` – označava da je neki postupak zastario (upozorenja pri kompilaciji)
 - ...
- Razred *MemberInfo* – (iz kojeg se izvodi `PropertyInfo` i slični)
 - `GetCustomAttribute` – vraća atribut danog tipa *Type* primijenjenog na asemblij, član razreda, ...
 - `GetCustomAttributes` – vraća polje atributa
 - `IsDefined` – određuje je li ijedan atribut zadanog tipa *Type* definiran

Definiranje naziva stupca atributom

- Umjesto naziva svojstva za naziv stupca uzima se vrijednosti atributa Display
 - Slično radi i MVC ako se koristi *label-for*
 - Primjer  MVC \ Extensions \ ExcelCreator.cs

```
public static ExcelPackage CreateExcel<T>(
    this IEnumerable<T> data, string worksheetName) {
    ...
    PropertyInfo[] props = typeof(T).GetProperties(...)
    foreach (var prop in props) {
        string name = prop.Name;
        if (prop.IsDefined(typeof(DisplayAttribute))) {
            name = prop.GetCustomAttribute<DisplayAttribute>().Name;
        }
        worksheet.Cells[row, col++].Value = name;
    }
}
```

Vlastiti atributi


- Vlastiti atributi nasljeđuju *System.Attribute*
- Naziv uobičajeno završava s *Attribute*, ali nije nužno
 - Ako završava s *Attribute*, taj se sufiks prilikom korištenja može se izostaviti

```
public class PrviAttribute : System.Attribute{
    public string Naziv { get; set; }
    public int Broj { get; set; }
}

public class DrugiAttribute : System.Attribute{
    public string X { get; set; }
}

,
[Prvi(Broj=5, Naziv="Test")]
[DrugiAttribute(X = "Proba")]
public class MojRazred{...}
```

Primjer vlastitog atributa

- Primjer  MVC \ Util \ ExcelFormatAttribute.cs
 - Atribut kojim će se definirati format ćelije u Excelu
 - Može se primijeniti samo na svojstva
 - određeno atributom *AttributeUsage*

```
[AttributeUsage(AttributeTargets.Property)]
```

```
public class ExcelFormatAttribute : Attribute {  
    public string ExcelFormat { get; set; } = string.Empty;  
  
    public ExcelFormatAttribute(string format)  
    {  
        ExcelFormat = format;  
    }  
}
```


Primjer korištenja vlastitog atributa (1)

- Primjer  MVC\ ViewModels \ ViewDokumentInfo.cs

```
public class DokumentViewModel {  
    ...  
    [ExcelFormat("0.00%")]  
    public decimal PostoPorez { get; set; }  
    [DisplayAttribute(Name = "Datum dokumenta")]  
    [ExcelFormat("dd.mm.yyyy")]  
    ...  
    [ExcelFormat("#,###,##0.00")]  
    public decimal IznosDokumenta { get; set; }  
}
```

- Napomena: EPPlus uvijek koristi engleske regionalne postavke za decimalni zarez odnosno točku

Primjer korištenja vlastitog atributa (2)

- Primjer  MVC \ Extensions \ ExcelCreator.cs

```
PropertyInfo[] props = typeof(T).GetProperties(...  
...  
foreach (T t in data) {  
    ++row; col = 1;  
    foreach (var prop in props)  
        ...  
        object value = prop.GetValue(t);  
        worksheet.Cells[row, col].Value = value;  
        if (prop.IsDefined(typeof(ExcelFormatAttribute))) {  
            string format = prop  
                .GetCustomAttribute<ExcelFormatAttribute>()  
                .ExcelFormat;  
            if (!string.IsNullOrEmpty(format)) {  
                worksheet.Cells[row, col]  
                    .Style.Numberformat.Format = format;  
            }  
        }  
    }  
}
```

Refleksija i performance

- Načelno, refleksija je spora. Npr. neka iz liste jednog tipa stvoriti listu drugog tipa, pri čemu tipovi imaju ista svojstva ili postoji uspostavljeno preslikavanje.
 - Kodu koji koristi biblioteku *AutoMapper* i preslikavanja korištenjem refleksija treba 5 puta više vremena od koda koji direktno kopira jedno po jedno svojstvo, a u nekim slučajevima (bez Automappera) to može biti i 45 puta sporije.
- Je li zbilja refleksija usko grlo? U slučaju upita na bazu, taj gubitak vremena je vjerojatno zanemariv u kontekstu cijelog postupka.

Donald Knuth: “Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.”

 - Promotriti projekt *ReflectionBenchmark* s rezultatima mjerenja