Fakulteta za Matematiko in Fiziko v Ljubljani

Abstraktni razred in Zbiralec smeti v C#

Predstavitev vprašanj pri predmetu Programiranje 3

Jakob Valič

Ljubljana, avgust 2018

Vsebina

Abstraktni razred v C#

Ključne besede: Razred, Dedovanje

Vir: http://www.csharpstar.com/abstract-vs-virtual-keyword-csharp/

Da bomo lažje razumeli abstraktne razrede, si najprej poglejmo, kako deluje dedovanje razredov. Če dedovanje razredov že razumemo, lahko skočimo na abstraktni razred.

Dedovanje razredov

Dedovanje razredov označujemo z operatorjem `:`. Imenujmo razred, ki deduje *podrazred* in razred od katerega deduje *nadrazred*. Zapis je sledeč: `[ime podrazreda] : [ime nadrazreda]`.

Ponazorimo to na sledečem primeru: Vsem umetnikom, naj so to glasbeniki, slikarji ali igralci je skupno, da se s svojo umetnostjo ukvarjajo določeno število ur dnevno. Prav tako imajo vsi nekje svoj atelje, studio ali glasbeno sobo, kjer ustvarjajo. Zato ustvarimo nadrazred `Umetnik`, ki hrani vse naštete podatke skupaj z metodo za opis umetnika.

```
class Umetnik
{
    protected string ime;
    protected string priimek;
    protected int stevilo_ur_dnevno;
    protected string kraj_delovanja;

    public Umetnik(string ime, string priimek, int stevilo_ur_dnevno, string kraj_delovanja)
    {
        this.ime = ime;
        this.priimek = priimek;
        this.kraj_delovanja = kraj_delovanja;
        this.stevilo_ur_dnevno = stevilo_ur_dnevno;
    }

    public string opisi_umetnika()
    {
        return String.Format("To je umetnik {0} {1}, ki deluje v kraju {2}. Ustvarja " +
            "povprečno {3} ur dnevno.",ime, priimek, kraj_delovanja, stevilo_ur_dnevno);
    }
}
```

Sedaj lahko ustvarimo primerek razreda Umetnik. Poglejmo izpis sledečih ukazov.

```
Umetnik Joško = new Umetnik("Jože", "Dolinšek", 5, "Črnomelj");
Console.Out.WriteLine(Joško.opisi_umetnika());
```

```
To je umetnik Jože Dolinšek, ki deluje v kraju Črnomelj. Ustvarja povprečno 5 ur
dnevno.
```

Recimo, da sedaj želimo razred, v katerem bi lahko povedali več o določenem tipu umetnikov. Naj bodo to slikarji. Ustvarimo podrazred `Slikar` ki bo od nadrazreda `Umetnik` podedoval vse njegove argumente (ime, priimek, kraj delovanja, čas ustvarjanja) in metode (opis umetnika). Dodali pa mu bomo metodo za opis tehnike, ki jo najraje uporablja umetnik, ki je slikar. Dedovanje označimo z `:`.

```
class Slikar : Umetnik
{
    private string tehnika;
    public Slikar(string ime, string priimek, int stevilo_ur_dnevno, string kraj_delovanja,
        string tehnika) : base(ime, priimek, stevilo_ur_dnevno, kraj_delovanja)
    {
        this.tehnika = tehnika;
    }

    public string opisi_tehniko()
    {
        return String.Format("Slikar {0} {1} najraje ustvarja svoje umetnine " +
            "s tehniko {2}.", ime, priimek, tehnika);
    }
```

Sedaj lahko ustvarimo primerek razreda `Slikar`. Uporabljamo lahko tako metode, ki se nahajajo v podrazredu `Slikar` kot tudi metode, ki jih je podrazred `Slikar` podedoval od nadrazreda `Umetnik`. Poglejmo izpis naslednjih ukazov.

```
Slikar Mare = new Slikar("Mare", "Gorišek", 5, "Jelenov Žleb", "voščenke");
Console.Out.WriteLine(Mare.opisi_umetnika());
Console.Out.WriteLine(Mare.opisi_tehniko());
```

```
To je umetnik Mare Gorišek, ki deluje v kraju Jelenov Žleb. Ustvarja povprečno 5
 ur dnevno.
Slikar Mare Gorišek najraje ustvarja svoje umetnine s tehniko voščenke.
```

Kaj je abstraktni razred?

Abstraktni razred v C# je osnova za izgradnjo podrazredov. Namenjen je le dedovanju. Vsebuje eno ali več abstraktnih metod.

Abstraktna metoda

Abstraktna metoda je oblike `abstract [tip, ki ga metoda vrne] [ime metode]([tipi in imena spremenljivk])`. Je navodilo ki zagotavlja, da bodo vsi podrazredi imeli definirano metodo s enakim imenom, enakim tipom, ki ga metoda vrne in enakimi tipi spremenljivk. V abstraktnem nadrazredu ta metoda nima telesa, saj je abstraktna. Če poskušamo dodati abstraktni metodi telo, nam Visual Studio javi sledeče opozorilo:

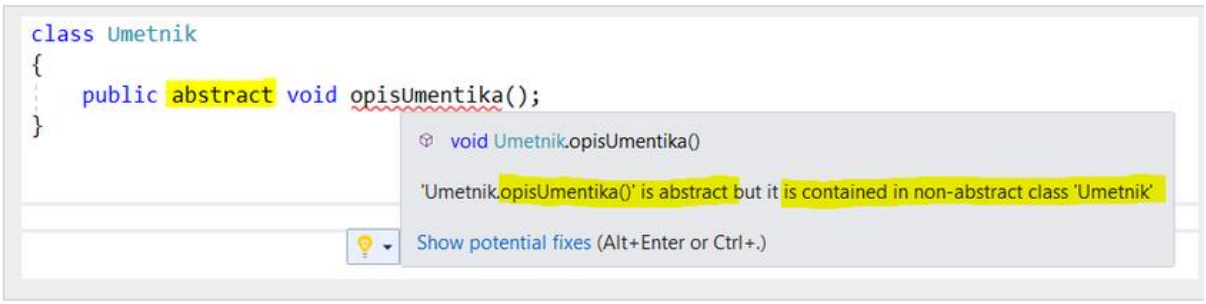
```
abstract class Umetnik
{
    public abstract void opisUmentika(){};
}
```

void Umetnik.opisUmentika()

'Umetnik.opisUmentika()' cannot declare a body because it is marked abstract

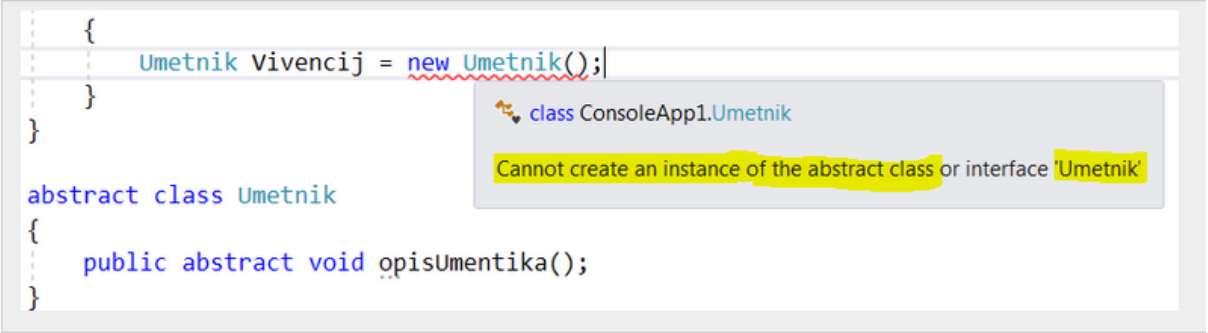
Show potential fixes (Alt+Enter or Ctrl+.)

Prav abstraktna metoda je tista, ki botruje temu, da označimo naš razred kot abstrakten. Če naredimo abstraktno metodo znotraj razreda, ki še ni označen kot abstrakten, dobimo sledeče opozorilo:



Ali lahko ustvarimo primerke abstraktnih razredov?

Primerkov abstraktnih razredov ne moremo ustvariti. Bistvo abstraktnih razredov je v tem, da so dodelani 'le na pol', saj služijo kot podlaga za ustvarjanje podrazredov. Visual Studio nam javi sledečo napako:



Dedovanje od abstraktnega razreda

Cilj vsakega abstraktnega razreda je v tem, da v svojem podrazredu doseže uresničitev. Ko podrazred podeduje od abstraktnega razreda, nas Visual Studio takoj spomni, da moramo dopolniti abstraktne metode, ki so napovedane v abstraktnem nadrazredu. Dobimo sledeče opozorilo:

Pravilni pristop je, da vse metode, ki so v originalnem razredu definirane kot abstraktne, povozimo (ang. override). V telesu določimo delovanje metode.

```
class Slikar : Umetnik
{
    public override void opisUmetnika()
    {
        Console.Out.WriteLine("To je umetnik, natančneje slikar.");
    }
}
```

Druga možnost pa je, da je tudi podrazred abstrakten. V tem primeru nam ni treba definirati abstraktnih metod, saj se te prenesejo. Na spodnjem primeru vidimo, da je podrazred `Glasbenik` prav tako abstrakten razred. Ker je podrazred, podeduje metodo `opisUmetnika`. Ni potrebno, da ji določi telo, ker je abstrakten. `Flavtist` je neposredo podrazred `Glasbenika` in preko `Glasbenika` tudi posredno podrazred `Umetnika`. Ker `Flavtist` ni abstrakten, mu je potrebno določiti metodo `opisUmetnika`. Hkrati kot zanimivost opazimo, da se da tudi v abstraktnem razredu narediti konstruktor. To še ne pomeni, da ga lahko ustvarimo kot primerek, temveč, da je v vseh podrazredih zahtevano podajanje argumenta za konstruktor, v našem primeru `string ime`. Če konstruktorja v abstraktnem razredu ne napišemo, ga Visual Studio avtomatično generira.

```
abstract class Umetnik
{
    public abstract void opisUmetnika();
}

abstract class Glasbenik : Umetnik
{
    protected string ime;

    protected Glasbenik(string ime)
        {
        this.ime = ime;
        }
}

class Flavtist : Glasbenik
{
    private string tip_flavte;

    public Flavtist(string ime, string tip_flavte) : base(ime)
    {
        this.tip_flavte = tip_flavte;
    }

    public override void opisUmetnika()
    {
        Console.Out.WriteLine(String.Format("To je flavtist {0}, " +
            "ki igra na {1} flavto.", ime, tip_flavte));
    }
}
```

Naslednji niz ukazov nam da vedeti, da smo pravilno uporabili abstraktne razrede:

```
Glasbenik Tino = new Flavtist("Tino", "prečno");
Tino.opisUmetnika();
```

```
To je flavtist Tino, ki igra na prečno flavto.
```

Lastnosti abstraktnih razredov

Povzemimo nekaj najpomembnejših lastnosti abstraktnih razredov:

- ne moremo ustvariti primerka abstraktnega razreda
- ustvarimo jih tako, da pred ime razreda dodamo besedico "abstract"
- abstraktni razred lahko deduje iz drugega abstraktnega razreda
- vsak razred, ki deduje od abstraktnega, mora implementirati vse abstraktne kompomente, razen če tudi sam ni abstraktnega tipa

Dodatne opombe:

Zanimiva je primerjava med abstraktnim razredom in vmesnikom (ang. interface). Medtem ko posamezen razred lahko deduje le od enega (abstraktnega) razreda, lahko pripada več vmesnikom.

Zbiralec smeti v C#

Ključne besede: Zbiralec Smeti

Vir:http://www.csharpstar.com/interview-questions-garbage-collection-csharp/, https://docs.microsoft.com

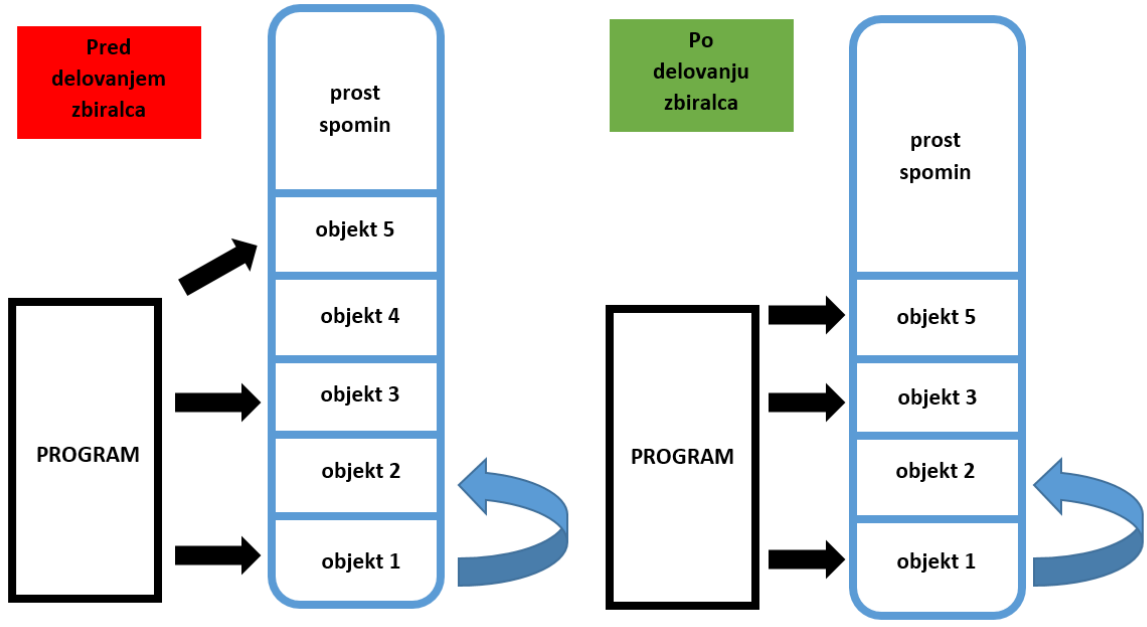
Zbiralec smeti (ang. garbage collector) je avtomatičen postopek, ki teče vzporedno z delovanjem programa. Namenjen je sprostitvi tistega dela delovnega spomina računalnika, v katerem se nahajajo objekti, do katerih program ne more več dostopati.

Sklad in kopica v pomnilniku

Pomnilnik je razdeljen na sklad in kopico. Na skladu se nahajajo prehodne spremenljivke, kot so npr. lokalne spremenljivke. Kopica hrani objekte. Za vsak objekt na kopici obstaja spremenljivka z referenco, ki kaže nanj. Po koncu delovanja določene funkcije, se vse njene lokalne spremenljivke zbrišejo s sklada. Pridemo do situacije, ko na kopici obstajajo objekti, na katere ne kaže nobena referenca več. Tu pride v igro zbiralec smeti. Proces zbiranja smeti se zažene tedaj, ko na kopici zmanjka prostora, da bi ustvarili nov objekt.

Kako zbiralec smeti deluje?

Ko se program, napisan v C# zažene, sistem določi nekaj spomina, da se program lahko izvaja. Tekom izvedbe program ustvari primerke različnih razredov, s katerimi upravlja. Na določeni točki program tega objekta več ne rabi. Tedaj program nima več dostopa do tega objekta. Tak objekt postane kandidat za zbiralca smeti. Poglejmo si spodnji primer, ki ponazori proces delovanja zbiralca smeti. Moder okvir ponazarja spominsko kopico računalnika. Vidimo, da pred delovanjem program kaže na objekte 1, 3 in 5. Ker na objekt 4 program ne kaže, ga zbiralec smeti pobriše. Zakaj ne pobriše objekta 2? Zato, ker nanj kaže objekt 1, na katerega imamo referenco.



Avtomatičen proces

Zbiralec smeti je avtomatičen proces. To je prednost, saj bi lahko pri ročnem sproščanju pomnilnika naredili katero od naslednjih napak:

- pozabili bi izbrisati objekt, na katerega ne kaže nobena referenca več,
- uničili bi lahko aktiven objekt, to bi vplivalo na vse objekte, ki imajo povezavo nanj,
- isti objekt bi lahko poskusili izbrisati večkrat.

Vseeno nam programski jezik C# omogoča, da sami naredimo razred, ki uporablja metode iz razreda `GC` (garbage collector). Naštejmo nekaj zanimivih metod `GC` razreda:

- `Collect()` Sproži proces zbiranja smeti na objektih vseh generacij.
- `Collect(Int32)` Sproži proces zbiranja smeti na objektih od generacije 0 do podane generacije.
- `GetTotalMemory(Boolean)` Vrne količino zasedenega pomnilnika v bytih. S parametrom dovolimo, da sistem konča s trenutnim procesom zbiranja smeti in nato vrne rezultat.
- `GetGeneration(Object)` Vrne trenutno generacijo določenega objekta.
- `KeepAlive(`Object`)` Doda referenco na objekt, kar zagotavlja, da ga zbiralec smeti ne bo počistil.
- `TryStartNoGcRegion(Int64)` Označuje začetek območja, na katerem je funkcija zbiralca smeti onemogočena. To je uporabno v izrednih primerih, ko bi zbiralec smeti s svojo dejavnostjo vplival na delovanje programa. Zbiralec smeti se vključi, če je medtem presežena podana največja količina spomina.
- EndNoGCRegion() Označuje konec območja, na katerem je funkcija zbiralca smeti onemogočena.

Generacije

Generacije so merska enota za določanje relativne starosti objektov v pomnilniku. Implementacija zbiralca smeti določa 3 generacije objektov, to so 0, 1 ali 2. Lastnost razreda `GC`, imenovana `MaxGeneration`, nam pove številko najvišje generacijo (privzeto 2).

Generacija 0 je najmlajša generacija. Vključuje kratkotrajne objekte, kot so na primer začasne spremenljivke. Zato je proces zbiranja smeti najpogostejši prav v tej generaciji. Vsi novi objekti najprej pridejo v generacijo 0, razen v primeru velikih objektov, ki so avtomatično preusmerjeni v generacijo 2.

Generacija 1 je prehodna generacija. Največ objektov iz generacije 0 sploh ne napreduje v to generacijo, ker so počiščeni. Tisti, ki napredujejo, pa imajo dobre možnosti za napredovanje v generacijo 2, ki vsebuje obstojne objekte. Ko zbiralec smeti pregleda to generacijo, avtomatično pregleda tudi generacijo 0 in 1.

Primer

Primer je slovenska kopija primera, ki se nahaja na https://docs.microsoft.com. Dobro prikazuje uporabo funkcij iz razreda `GC`.

```
class MojZbiralecSmeti
{
    private const long števec = 1000;

    static void Main()
    {
        MojZbiralecSmeti mojZbiralec = new MojZbiralecSmeti();

        Console.WriteLine("Najvišja generacija je {0}", GC.MaxGeneration);

        mojZbiralec.UstvariSmeti();
        mojZbiralec.IzpišiGeneracijo();
        mojZbiralec.IzpišiZasedenSpomin();
        GC.Collect(0); // Počistimo le generacijo 0
        mojZbiralec.IzpišiGeneracijo();
        mojZbiralec.IzpišiZasedenSpomin();
        GC.Collect(2); // Počistimo generacije 0, 1, 2
        mojZbiralec.IzpišiGeneracijo();
        mojZbiralec.IzpišiZasedenSpomin();
        Console.Read();
    }

    void IzpišiGeneracijo()
    {  Console.WriteLine("Generacija: {0}", GC.GetGeneration(this));   }

    void IzpišiZasedenSpomin()
    {  Console.WriteLine("Zaseden spomin: {0}", GC.GetTotalMemory(false));  }

    void UstvariSmeti()
    {
        Version vt;
        for (int i = 0; i < števec; i++)
        {
            vt = new Version();
        }
    }
}
```

Ko poženemo program, dobimo sledeč izpis. Izpis nam pove, da je na začetku objekt `mojZbiralec` pripadal generaciji 0. Poklicali smo zbiralca smeti, ki je sprostil približno tretjino delovnega spomina. Metoda `UstvariSmeti` namreč ustvarja objekte, na katere ne kaže nobena referenca. Po zbiranju smeti je naš objekt prešel v generacijo 1. Sedaj pokličemo čiščenje vseh treh generacij. Tokrat se ne sprosti skoraj nič dodatnega spomina, vsi nerabljeni objekti so namreč bili počiščeni že v generaciji 0 in niso napredovali v generacijo 1. Ker je objekt prestal to čiščenje, ga `GC` promovira v generacijo 2.

```
Najvisja generacija je 2
Generacija: 0
Zaseden spomin: 62732
Generacija: 1
Zaseden spomin: 40356
Generacija: 2
Zaseden spomin: 40280
```