C# Životni ciklus objekta

Katedra za informatiku
Fakultet tehničkih nauka
Univerzitet u Novom Sadu

Kreiranje objekta

- Kreiranje objekta

```
Racunar r = new Racunar();
```

1. CLR alocira jedan deo heap memorije za objekat
   - Nije pod kontrolom programera
2. Inicijalizuje se objekat
   - Pod kontrolom programera kroz kod u konstruktoru

Uništavanje objekta

- Garbage Collector
  - deo CLR koji se povremeno aktivira i uništava objekte koji se ne koriste
  - Ne može programer da inicira uništavanje objekta
- Uništavanje se sastoji od
  1. Izvršavanja destruktora
     - Nekad je potrebno izvršiti programski kod nakon uništavanja objekta da bi sistem nastavio ispravan rad
     - Programer piše destruktor - ne možemo biti sigurni da će se kod izvršiti jer to zavisi od toga da li će biti potrebe da se aktivira Garbage collector
  2. Dealociranja memorije
     - CLR oslobađa memoriju koja je bila zauzeta od strane objekta

Destruktor

- Nasleđe C++ jezika gde se oslobađanjem memorije bavio programer
- Najčešće je nepotreban obzirom da CLR automatski uništava objekat
  - Koristan je nekad zbog optimizacije da bi se efikasnije uništio objekat
  - Ili da bi se zatvorila veza ka bazi ili fajlu (mada ima i boljih načina da se ovo uradi)

Destruktor sintaksa

```
class FileProcessor
{
  FileStream file = null;
  public FileProcessor(string fileName)
  {
      this.file = File.OpenRead(fileName);
  }
  ~FileProcessor()
  {
      this.file.Close();
  }
}
```

Ograničenja za destruktor

- Može se definisati samo za adresne tipove
- Ne može se direktno pozvati iz koda i zato ne može
  - da ima modifikator pristupa (npr. **`public`**)
  - pa prima parametre (npr. **`~FileProcessor(int x)`**)

Implementacija destruktora

- Kompajler automatski prevodi destruktor u metodu *Finalize* nasleđenu od klase *Object*

```
protected override void Finalize()
{
  try { //programski kod iz destruktora }
  finally { base.Finalize(); }
```

- Ne možemo napisati svoju redefiniciju metode *Finalize*
- Ne možemo direktno pozvati metodu *Finalize*

Garbage Collector - aktiviranje

- GC se aktivira samo kada ima potrebe
  - Kada količina heap memorija postane premala
  - Dakle, ne aktivira se odmah kad se objekat više ne upotrebljava
- GC uništava samo objekte na koje ne postoji nijedna aktivna referenca
- Eksplicitni poziv Garbage collectora
  - System.GC.Collect()
  - Ne preporučuje se ovakav eksplicitni poziv

Garbage Collector - implementacija

- Izvršava se kao posebna nit
  - odvojen tok izvršavanja koji se izvršava konkurentno sa glavnim tokom izvršavanja
- Aktivira se periodično
- Kada se aktivira, ostale niti u aplikaciji su blokirane jer GC premešta objekte u memoriji
- Održava mapu dostupnih objekata
- Svi koji nisu u ovoj mapi smatraju se nedostupnim i uništavaju se
  - Pre toga se pozivaju njihovi destruktori ako postoje

Disposal metoda

- Gde objekat treba da obavi završne operacije?
  - Npr. zatvaranje veze ka fajlu

```
reader.Close();
```

  - U destruktoru je nepouzdano
    - Ne znamo da li će se i kada pozvati
  - Na kraju metode nije *exception-safe*
    - Ako se u toku metode desi izuzetak kod se neće izvršiti
  - U *finally* bloku je pouzdano, ali komplikuje kod

Disposal metoda

- Standardan i najbolji način je korišćenjem *using* bloka

```
using (TextReader reader = new StreamReader(filename))
{
      string line;
      while ((line = reader.ReadLine()) != null)
      {
            Console.WriteLine(line);
      }
}
```

- Objekat će biti uništen kada se završi *using* blok

IDisposable interfejs

- U *using* izrazu može da se koristi samo objekat koji implementira *IDisposable* interfejs
- Interfejs ima samo jednu metodu

```
void Dispose();
```

- Na kraju *using* bloka automatski će biti pozvana *Dispose* metoda
- U ovoj metodi je potrebno napisati završne operacije koje objekat treba da obavi pre uništavanja
  - Npr. u *TextReader* klasi je u ovoj metodi zatvorena veza ka fajlu