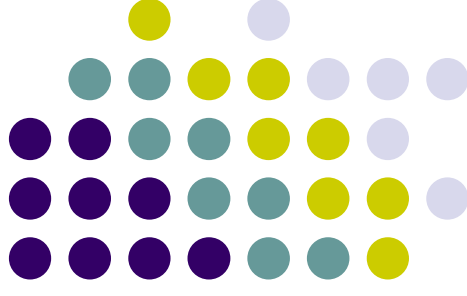
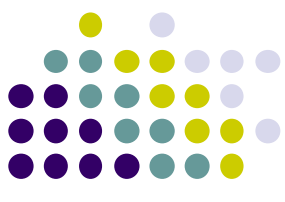


Distribuirani procesi

Vježbe 03

Vinko Petričević

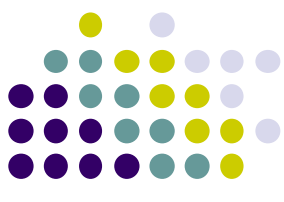




Exceptions

```
try { ... }  
catch (tipExceptiona1 e) { ... }  
catch (tipExceptiona2 e) { ... }  
catch { ... }
```

- exceptioni su izvedeni iz `System.Exception`
- exception slijedi `call-context` (logičku dretvu), tj. ako nije uhvaćen na serveru, treba ga proslijediti klijentu koji je pozvao funkciju. Obični `Exception` bi srušio server
- U `remotingu` se šalju po vrijednosti, i trebaju biti izvedeni iz `System.Runtime.Remoting.RemotingException`



Exceptions

```
[Serializable]
public class CustomRemotableException : RemotingException, ISerializable
{
    private string _internalMessage;

    public CustomRemotableException() { _internalMessage = String.Empty; }

    public CustomRemotableException(string message) { _internalMessage = message; }

    public CustomRemotableException(SerializationInfo info, StreamingContext context) {
        _internalMessage = (string)info.GetValue("_internalMessage", typeof(string));
    }

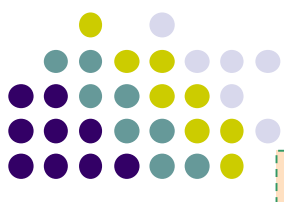
    public override void GetObjectData(SerializationInfo info, StreamingContext context) {
        info.AddValue("_internalMessage", _internalMessage);
    }

    // Returns the exception information.
    public override string Message {
        get { return "Ovo je iznimka: \'" + _internalMessage + "\'"; }
    }
}
```



Slanje objekata po referenci

- Kada na klijentu kreiramo objekt sa servera, svi pozivi funkcija i pristupi svim elementima objekta se marshaliraju i izvršavaju u serverskoj aplikacijskoj domeni.
- Ako želimo neku klasu poslati po referenci na server (ili promijeniti element koji je referenca), na serveru trebamo podesiti `Security`
- To se može pomoću `app.config` datoteke ili programski, na sljedeći način:



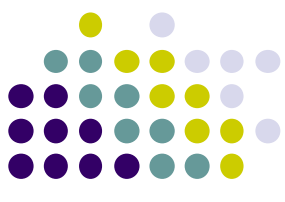
Slanje objekata po referenci

```
using System.Runtime.Remoting.Channels;
using System.Collections;
...
BinaryServerFormatterSinkProvider provider =
    new BinaryServerFormatterSinkProvider();
provider.TypeFilterLevel =
    System.Runtime.Serialization.Formatters.TypeFilterLevel.Full;

IDictionary props = new Hashtable();
props["port"] = 12345; // ili 0 za klijenta

TcpChannel chan = new TcpChannel(props, null, provider);
...
```

- Ako želimo da server može pozivati funkcije na klijentu, istu stvar treba napraviti i na klijentu (i objekt mora biti izveden iz `MarshalByRefObject`)



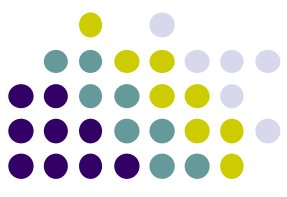
Slanje objekata po referenci

- Ako neki objekt pošaljemo po referenci kao parametar funkciji od serverske klase, svi pozivi klijentskom objektu se marshaliraju i izvršavaju u klijentovom procesu/računalu
- Ako ga server prosljedi nekom trećem objektu na nekom trećem procesu, sve će analogno raditi (ukoliko taj treći može vidjeti klijenta – ako su u istoj mreži)
- Ista stvar se događa i ako neki element servera promjenimo na neki klijentski objekt



Zadatak – primjer 3.1

- Napravite jednostavan chat server
- Na server se može spojiti više klijenata
- Klijent šalje poruku, a server ih pošalje svim ostalim klijentima
- Neka server vodi računa o tome da je neki klijent možda ‘nestao’



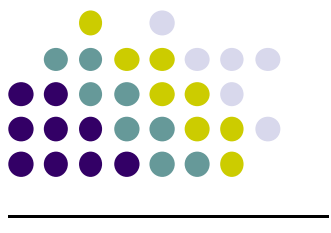
Primjer 3.1

- da ne bismo komplicirali s 2 projekta, potrebno je napraviti samo jedan C# Console Application
- U reference dodati System.Runtime.Remoting
- Pokrenuti ga 2 puta. Prilikom prvog pokretanja, rezervirat će se port 12345, te će se startati server
- Prilikom drugog pokretanja, nakon neuspješne rezervacije porta 12345, pokrenut će se klijent



.net eventi – primjer 3.2

- EventWaitHandle interface (AutoResetEvent, ManualResetEvent) i WaitHandle (Mutex, Semaphore) su izvedeni iz MarshalByRefObject, te se mogu koristiti pri signalizaciji



Zadatak

- Napravite jednostavnu klijent/server aplikaciju
- Server po primitku poruke signalizira event klijenta
- Klijent ima 2 threada. Jedan svakih 5 sekundi (ili ranije ako dobije signal) ispiše poruku, a drugi thread učitava poruku s komandne linije, te ju pošalje serveru



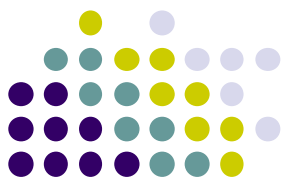
CallContext – primjer 3.3

- Za razmjenu podataka među procesima možemo (osim .net remotinga) koristiti CallContext podatke (jer u .net-u ne postoji globalna memorija)
- `GetData(naziv_varijable)` daje određeni podatak
- `SetData(naziv_varijable, vrijednost)` postavlja određeni podatak. Vrijednost mora biti tip izveden iz `ILogicalThreadAffinative` kojeg je moguće serijalizirati



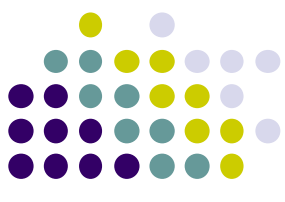
ContextBoundObject

- Svaka aplikacija sadrži jedan ili više konteksta (`System.Runtime.Remoting.Contexts.Context`)
- `Context.DefaultContext`
- `Thread.CurrentContext`
- threadovi mogu ući i napustiti kontekst po želji
- Ako tip izvedemo iz `ContextBoundObject`, on će uvijek živjeti u kontekstu u kojem je kreiran



LifeTime – primjer 3.4, 3.5

- U .net-u se sve klase ponašaju kao pametni pokazivači. Tj. Kada im broj referenci padne na 0, prosljede se GC-u koji će ih uništiti kada mu bude najbolje odgovaralo
- Objekti na udaljenim računalima se ne uništavaju odmah.
- distributed GC koristi LifetimeServices da odredi nakon koliko vremena treba provjeriti mogu li se objekti uništiti
- **I oni će biti uništeni ako određeno vrijeme nije bilo ni jednog poziva objektu, iako im je broj referenci veći od 0 – da bi se rasteretila mreža i zbog mogućih grešaka u komunikaciji.**
- Moguće je dodati Sponzora, kojeg će se nakon isteka vremena prvo pitati može li se objekt uništiti – primjer 3.5



Delegates – primjer 3.6

- Možemo ih shvatiti kao interface s jednom metodom
- Osim sinhronog pozivanja funkcije (Invoke) omogućavaju i asinhrono (BeginInvoke, EndInvoke) koje se automatski odvija u zasebnom threadu
- Dakle ovaj primjer prije spada u 1. vježbe