8. Proceduralne apstrakcije

Dve važne vrste apstrakcija koje podržavaju programski jezici su:

- Proceduralne apstrakcije
- Apstrakcije podataka

- Stariji koncept u programskim jezicima
- Koncept je podržan u gotovo svim višim programskim jezicima preko *potprograma*
  - apstraktni mehanizam koji omogućava višestruko korišćenje jednom napisanog koda (code reuse)
  - podržava modularnost

Potprogram

- *inkapsulira* algoritam i lokalne promenljive.
  inkapsulacija: odvajanje specifikacije potprograma od njegove implementacije
- Korisnik koristi potprogram preko interfejsa potprograma bez poznavanja detalja kako je potprogram implementiran (*skrivanje informacija*).
- *Primer:* Poziv potprograma za sortiranje numeričkih vrednosti
  sort (numbers, n) // call sort
  je apstrakcija algoritma za sortiranje

Potprogram

- podržava modularnost
- omogućava višestruko korišćenje koda (code reuse)
- Generalno postoje dve vrste podprograma
  - **Procedure** su kolekcija instrukcija koja definišu parametrizovana sračunavanja.
  - **Funkcije** (model mat. funkcije) - semantički modeli matematičkih funkcija
    - u praksi mnoge funkcije ponašaju se kao procedure
- *primer* funkcije u pj C:
- void sort (int list [ ], int listlen);

Definicija potprograma

- **Zaglavlje potprograma** - opisuje interfejs potprograma
- **Telo potprograma** - sadrži deklaracije lokalnih promenljivih i algoritam

Zaglavlje potprograma specificira

- da je sintaksni unit definicija potprograma i vrstu potprograma
- ime potprograma
- listu parametara (opciono)

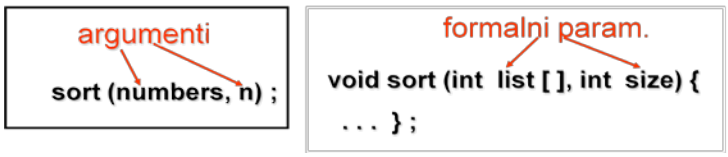
Primer zaglavlja potprograma u pj Ada: **procedure** brojac (< parametri >)
Primer zaglavlja u pj C: **void** brojac ((< parametri >)

Signatura potprograma def. broj i tipove formalnih parametara, obično i redosled

- *formalni parametrar* sintaksno je tzv. “dummy” promenljiva

Poziv potprograma eksplicitno se zahteva izvršavanje potprograma

Stvarni parametri (argumenti) reprezentuju vrednosti ili adrese koje se koriste u izrazu za poziv potprograma (konstante, promenljive ili izrazi)



Povezivanje formalnih i stvarnih parametara u vreme poziva potprograma realizuje se na osnovu

- Pozicije (pozicioni parametri)
- Parametri bazirani na ključnoj reči
- "Default" vrednost za parametar

- **Pozicioni parametri**
  - u većini PJ parametri su pozicioni
  - redosled povezivanja je bitan (prvi argument povezuje se sa prvim parametrom, itd.)

Primer:

```
void sort (int list [ ], int size) {          // definicija potp. sort
            . . .
};
sort (numbers, n) ;              // poziv sort
```

- **Parametri bazirani na ključnoj reči**
  - redosled je irelevantan
  - ime formalnog parametra povezuje sa imenom argumenta
  - Ada

Primer:

```
void sort (int list [ ], int size) {              // definicija pot. sort
            . . .   } ;
sort (size -> n, list ->numbers);              // poziv sort
```

- **"Default" vrednost za parametar**

  Primer:

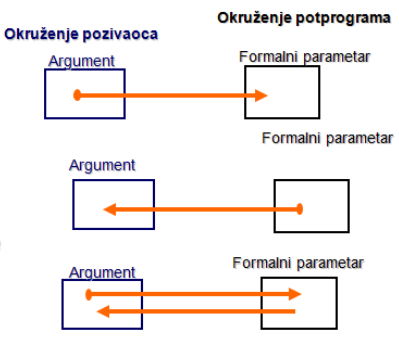
```
void sort (int list [ ], int size = 50) ; // def. sort
sort (numbers) ;  // poziv sort
```

  - Ada, C++

Semantički modeli za prenos parametara

način na koji se parametri prenose u ili iz pozvanog potprograma

- **In mode**
- **Out mode**
- **InOut mode**



Implementacioni modeli prenosa parametara

- Pass-by-value (in)
- Pass-by-result (out)
- Pass-by-value-result (in out)
- Pass-by-reference (inout)

Metode prenosa parametara

Pass-by-value (in)

- Vrednost argumenta kopira se u formalni parametar (inicijalizacija formalnog parametra)
- Formalni parametar posmatra se kao lokalna promenljiva za sve vreme izvršavanja potprograma

- Implementacija: kopiranje vrednosti
  - cena operacija kopiranja i memorisanja u slucaju kada je parametar velikih dimenzija (na primer niz sa velikim brojem elemenata)
- Jednostavna metoda, koristi se u velikom broju programskih jezika
  C, Pascal, Modula-2, C++, Java, C#

Pass-by-value *primer u C#*

```
class  ClassA  {
        public int  ChangeValue ( int v )
        {       v = v  + 1;  return  v ; }
}
class PassByValueTest  {
        static void Main ()
        {       ClassA ca = new ClassA() ;
                int  v1 = 8 ;
                int  v2 = ca.ChangeValue (v1);
                System.Console.WriteLine ("v1 = {0}, v2 = {1}", v1, v2);
        }
}
```

Pass-by-result (out)

- Kada potprogram završi sa radom, vrednost formalnog parametra prenosi se (kopira) u odgovarajući argument
- Parametar se tretira u potprogramu kao lokalna promenljiva i nema inicijalizacije preko vrednosti argumenta
- Metoda je uvedena u programskom jeziku ALGOL 60
  OUT parametar u C#

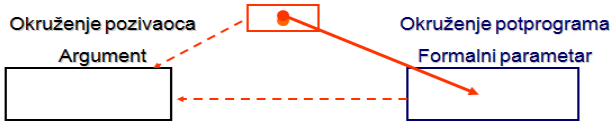
Pass-by-result (out) *primer u C#*

```
class  ClassA  {
        public void  SetValue( out  int  v )
        {       v = 10 ;  }
}
class PassByResultTest  {
        static void Main ()
        {       ClassA ca = new  ClassA() ;
                int  v1 ;
                ca.SetValue ( out  v1 ) ;
                System.Console.WriteLine ("v1 = {0}", v1 ) ;
        }
}
```

Pass-by-value-result (in out)

- Kombinacija metoda pass-by-value i pass-by-result
- Zahteva dve operacije kopiranja
- Korišćen u ALGOL-W, kasnijim verzijama FORTRAN-a, Ada

Pass-By-Reference



- Adresa argumenta prenosi se u formalni parametar
- Formalni parametar i referenca na argument su alijasi
- Efikasnnost (prosta implementacija, nema dupliciranja memorije i kopiranja)
- Sporiji pristup (u odnosu na metodu pass-by-value)
- Formalni parametri i argumenti su alijasi
- C++, Pascal, Modula-2, Java, C# (parametar *ref*)

Pass-By-Reference *primer u C#*

```
class  ClassA  {
        public void  SetValue( ref  int  v )
        {       v = 10 ;  }
}
class PassByResultTest  {
        static void Main ()
        {       ClassA ca = new  ClassA() ;
                int  v1 = 0 ;
                ca.SetValue ( ref  v1 ) ;
                System.Console.WriteLine ("v1 = {0}", v1 ) ;
}}
```

9. Apstrakcije podataka

Koncept apstraktnog tipa podatka

- Tip je *apstraktan*, ako su klijentima (programima koji koriste definisani tip) vidljive samo
  - **Skup** (apstraktnih) **operacija** preko kojih se stanja objekata tipa jedino mogu menjati – *javni interfejs ATP*
  - **Naziv tipa** - Klijenti mogu da kreiraju instance (objekte) definisanog ATP

- Primitivni tipovi podataka kao apstraktni tipovi (na primer int type)
  - za korisnike primitivnog tipa vidljivi su: *ime tipa* i *skup operacija* za manipulisanje vrednostima tipa;
  - u korisničkim programima deklariše se promenljiva tipa **int**
  - interna reprezentacija vrednosti tipa i implementacija operacija tipa su skrivene za korisnike

Apstraktni tip podatka je korisnički definisan tip koji zadovoljava sledeća dva uslova:

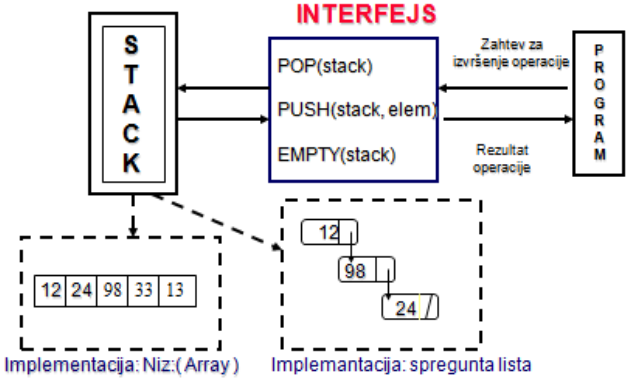
- **Inkapsulacija** (Encapsulation)
- **Skrivanje informacija** (Information hiding)

- **Inkapsulacija:** odvajanje specifikacije ATP od njegove implementacije
  - Deklaracija tipa i operacija nad objektima tipa definišu se u jednoj sintaksnoj konstrukciji
  - Reprezentacija tipa i implementacija njegovih operacija definišu se u istom ili posebnom sintaksnom unit-u; u oba slučaja implementacija mora biti skrivena za klijente
  - Drugim programima je dozvoljeno da kreiraju promenljive (objekte) definisanog tipa i da koriste javni interfejs ATP

Prednosti inkapsulacije:

- Pakovanje deklaracija tipa i njegovih operacija u jedan sintaksni unit obezbenuje se metod za organizovanje programa u logičke unit-e koji se mogu nezavisno kompajlirati.
- Pakovanje implementacije tipa i njegovih operacija u zaseban sintaksni unit je dobar način za razdvajanje specifikacije od implementacije
- Ako su i deklaracija i implementacija tipa i njegovih operacija u istom sintaksnom unit-u onda mora postojati neki način za sakrivanja implementacije tipa od klijenta

- ***Skrivanje informacija:*** omogućava korisćenje interfejsa ATP bez poznavanja njegove implementacije
- Prednosti skrivanja informacija
  - Promena implementacionih detalja ATP, ima samo *lokalni* efekat (ne zahtevaju se izmene u klijentskim programima)
  - Sigurnost – klijenti ne mogu direktno da menjaju osnovnu reprezentaciju objekata ATP
  - Primeri ATP: stack, red, stablo, graf



Primer korisnički definisanog ATP Stack
Stack ima sledeće apstraktne operacije:

- create(stack)
- destroy (stack)
- empty (stack)
- push (stack, element)
- pop (stack)
- top (stack)

Klijent ATP Stack ima sledeći kod:

```
. . .
create(stck);
push(stck, 5);
push(stck, 8);
pop(stck);
```

Podrška definisanju apstraktnog tipa podatka u programskim jezicima

Programski jezik podržava ATP ako sadrži:

- sintaksni unit za inkapsulaciju implementacije ATP (reprezentaciju tipa i implementaciju njegovih operacija)
- mehanizam koji treba da obezbedi vidljivost (za klijente) javnog interfejsa ATP. Interfejs omogućava klijentu da deklariše instance ATP i da manipuliše njihovim vrednostima.

Primeri ATP u programskim jezicima: Ada, C++, Java, C#

ATP u programskom jeziku Ada

- **Inkapsulacija** - inkapsulacioni mehanizam je **package** koji se sastoji iz dva dela:
  - **Specification package** (javni interfejs)
  - **Body package** (implementacija imenovanih entiteta u specifikaciji)
  - mogu se odvojeno kompajlirati
- **Skrivanje informacija** - reprezentacije tipa obezbeneno je pomoću dve sekcije u paketu za specifikaciju – jedna sekcija u kojoj su entiteti vidljivi za klijente, a druga sakriva svoj sadržaj.
  Reprezentacija tipa nalazi se u privatnom delu (koristi se **private** klauzula) paketa za specifikaciju (specification package)

ATP nazvan Stack_Type deklarisan je u vidljivom delu paketa za specifikaciju: **Type** Stack_Type **is private**;
Kompletna definicija tipa Stack_Type data je u privatnom delu (nevidljivom za klijenta) paketa za specifikaciju koji je označen sa klauzulom **private**

```
package … is
        type Stack_Type is private;
                …
        private
                type NODE_TYPE is
                        record
                        …
                        end record;
                …
```

Razlog zadavanja reprezentacije tipa u paketu za specifikaciju: Reprezentacija tipa nije vidljiva za klijenta, ali mora biti vidljiva za kompajler (klauzula **private** je vidljiva za kompajler)

Ada Stack_Pack ATP : specification package

```
package Stack_Pack  is
-- javni interfejs
    type Stack_Type is private;
        function Empty (Stk : in Stack_Type)  return Boolean;
                        Elem : in Integer);
        procedure Push(Stk : in out Stack_Type; Elem : in Integer);
    procedure Pop (Stk : in out Stack_Type);
    function Top (Stk : in Stack_Type)  return  Integer);
```

```
    private -- deo koji je skriven od  klijenata
        Max_Size : constant  :=  100;
        type  List_Type is array( 1..MaxSize) of Integer;
         type Stack_Type is
         record
            List : List_Type;
           TopIndex :Integer range  0..Max_Size  := 0;
        end record;
end Stack_Pack;
```

Korišćenje ADA Stack_Pack ATP

```
with  Stack_Pack,  Ada.Text_IO;
use  Stack_Pack,  Ada.Text_IO;
Procedure  Use_Stack  is
        TopElem : Integer;
        Stack : Stack_Type;
        Push(Stack, 40);
        Push (Stack, 20);
        TopElem := Top(Stack);
        Pop(Stack);
. . .
end Use_Stack;
```

Ada Stack_Pack ATP: body package

```
with Ada.Text_IO; use  Ada.Text_IO;
package body  Stack_Pack   is
   function  Empty (Stk : in Stack_Type)  return
Boolean  is
     begin
          return Stk.TopIndex = 0;
    end Empty;
   procedure  Push(Stk : in out Stack_Type; Elem :
in Integer)  is
     begin
          if  Stk.TopIndex >= Max_Size  then
            Put_Line("ERROR- stack overflow");
          else
            Stk.TopIndex := Stk.TopIndex + 1;
            Stk.List(TopIndex) := Elem;
          end if;
     end Push;
   procedure Pop (Stk : in out Stack_Type)  is
     begin  . . .  end Pop;
   function Top (Stk : in Stack_Type) return
Integer)  is
     begin  . . .  end Pop;
end Stack  Pack;
```

- Klijenti mogu da pristupe nekoj od javnih članica Ada package direktno pomoću njegovog imena;
- Klauzula **with** obezbeđuje vidljivost imena definisanih u eksternim paketima
- Klauzula **use** eliminiše eksplicitnu kvalifikaciju referenci na imena definisana u package (dato u primeru)

10. Apstrakcije podataka

Primeri definisanja apstraktnog tipa podatka u pojedinim programskim jezicima

ATP u programskom jeziku C++

- **Inkapsulacija**

  ATP se implementira korišćenjem **klase**
  - Članice (podaci i funkcije) definisane su u klasi
  - Instance (objekti) klase mogu biti stack_ dinamičke ili heap-dinamičke
  - Eksplicitna alokacija (**new**) i dealokacija (**delete**) heap-dinamičkih objekata

- **Skrivanje infromacija**
  - **private** klauzula (entiteti u private klauzuli su vidljivi samo za članice klase i friends klase)
  - **public** klauzula (javni entiteti su u public klauzuli; ova klauzula opisuje javni interfejs za objekte klase)
  - **protected** klauzula za nasleđivanje

Razlika u podršci ATP C++ i Java:

- U C++, klijenti mogu pristupati nekoj od javnih članica klase, ali samo preko instanci (objekata) klase.
- Klijentski programi mogu da pristupe nekom od javnih entiteta Ada package direktno pomoću njegovog imena

C++ ATP stack

```
#include <iostream.h>
class stack {
  private: // vidljive za članice klase
   int *stackPtr;
   int maxLen;
   int topPtr;
  public: // vidljive za klijente
   stack() { // konstruktor
            stackPtr = new int [100];
            maxLen = 99;  topPtr = -1; }
   ~stack() { delete [ ] stackPtr; } // destruktor
   void push(int number) { ... }
   void pop( ) {  ... }
   int top( ) { return (stackPtr[topPtr]) };
   int empty() { topPtr == -1;}
}
```

Koriscenje C++ ATP stack

```
void main( ) {
  int  top;
 stack stk;
  stk.push (42)
  stk.push (20);
  top = stk.top( );
  stk.pop( );
  . . .
}
```

ATP u programskom jeziku Java

- **Inkapsulacija**
  - Svi korisnički definisani tipovi su **klase**
  - Svi objekti su heap-dinamički
  - implicitna dealokacija
- **Skrivanje informacija**

Za svaku članicu klase definiše se vrsta pristupa; koriste se ***modifikatori pristupa*** umesto klauzula:

- **public**
- **private**
- **protected**
- **internal**
- **protected internal**

Java StackClass ATP

```
import java.io.*;
class StackClass {
 private int [ ] stackRef;
 private int maxLen, topIndex;
 public StackClass( ) { //Konstuktor
     stackRef = new int [100];
     maxLen = 99;
     topIndex = -1;
 }
 public void push(int number) {
     if (topIndex == maxLen)
             System.out.println ("Error: stack is full");
     else  stackRef [++topIndex] = number ;
}
 public void pop() {  // brise broj na vrhu staka
     if (topIndex == -1)
             System.out.println ("Error: stack is
     empty");
     else  --topIndex;
 }
 public int top() { return (stackRef [ topIndex ]); }
 public boolean empty() { return (topIndex == -1);}
}
```

Korišćenje

```
public class TestStack {
  public static void main(String[] args ) {
     StackClass myStack = new StackClass ();
     myStack.push (40):
     myStack.push (20);
     mystem.out.println ("  " + myStack.top());
     myStack.pop ();
     System.out.println ("  " + myStack.top());
     myStack.pop ();
     myStack.pop ();  // error!!!
  }
}
```

- pristup *privatnim* članicama-podacima klase: metode pristupa (**get** i **set**)
- C# obezbeđuje ***properties*** kao način implementacije metoda pristupa (set i get); ne zahteva se eksplicitni poziv metoda

U klasi Student definisan je property Year, koji omogućava metodama get i set pristup do privatne članice-podatka: year

```
public class Student {
        public int Year  {  //* Year je property
                    get  { return year; }
                    set  { year = value; }
        }
        private int  year;
        . . . .
} ….
        Student s = new  Student( );
        s.Year = 1;
        s.Year = s.Year + 1;
```

Složeni inkapsulacioni mehanizmi

Inkapsulacione konstrukcije

- U prethodnom delu razmatrane su sledeće (minimalne) inkapsulacije
  - Package u Adi
  - Klase u C++, Javi i C#
- Kod velikih programa, dva problema postaju evidentna:
  - logička organizacija i
  - rekompilacija
- Rešenje za oba problema:
  - grupisanje programa u kolekcije logički povezanog koda, gde se svaka od njih može pojedinačno kompajlirati (rekompajlirati
  - Takve kolekcije nazivaju se **inkapsulacione** (*inkapsulacion)* **konstrukcije**

Inkapsulacione konstrukcije - sintaksni kontejneri za logički povezane softverske resurse, gde se svaki od njih može kompajlirati pojedinačno

- **Ugnježdavanje potprograma**
  - programi se organizuju pomoću ugnježdavanja definicija potprograma unutar većih potprograma u kojima se koriste
  - Ovaj metod organizovanja programa koristi statički doseg i nije primarna inkapsulaciona konstrukcija koju koriste savremeni pj
  - Ada, Pascal, FORTRAN 95
- **Inkapsulacione konstrukcije u C**
  - Ne pruža strogu podršku za inkapsulaciju ali se može simulirati
  - kolekcija povezanih funkcija i podataka smešta se u **implementacionu datoteku**, koja se može nezavisno kompajlirati
  - *Interfejs* impl. datoteke smešta se u posebnu datoteku, tzv. **header file** *(* **.h file** *). Header file* sadrži sadrži deklaracije funkcija i tipova.
  - header file se uključuje u klijentski kod (pomoću **#include** specifikacije)
  - Ovaj pristup omogućava odvajanje specifikacije od implementacije

Primer:
Header datoteka convert.h

```
#define FACTOR ((double) 1.609344)
double MileToKilometers( double miles );
```

implementaciona datoteka convert.c

```
#include “convert.h"
double MileToKilometers( double miles ) {
return FACTOR * miles;
}
```

- **Inkapsulacije konstrukcije u C++**
  - slično kao u C-u, sa header datotekama za razdvajanje specifikacije od implementacije
  - Problem pristupa neke operacije objektima dve klase. U C++ uvedene **friends** funkcije i klase
  - C++ dozvoljava da klase deklarišu druge klase ili funkcije kao “**friends”**
  - “**friends”** funkcije i klase imaju pristup privatnim članicama klase u kojoj su deklarisane kao **friend**

Primer korišćenja friend funkcije za množenje matrice i vektora

```
class Vector {
        friend Vector multiply(const Matrix&, const Vector&);
        ...
};
class Matrix {
        friend Vector multiply(const Matrix&, const Vector&);
        ...
};
// funkcija multiply definisana izvan definicija dve klase klasa
Vector multiply(const Matrix& m1, const Vector& v1){
        ...
        };
```

- **Inkapsulacione konstrukcije u programskom jeziku Ada**
  - inkapsulaciona konstrukcija **package** koji može da sadrži više ATP-a

Ada packages

inkapsulaciona konstrukcija **package** koji može da sadrži više ATP-a

C# Assemblies

veća konstrukcija od klase i koristi se od strane svih .NET programskih jezika

- **Inkapsulacije konstrukcije u programskom jeziku C#**

– **Assemblies,** veća konstrukcija od klase i koristi se od strane svih .NET programskih jezika
– Assembly se kreira u procesu kompilacije c# programa

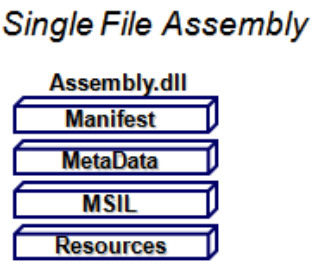
– Assembly je kolekcija datoteka koje mogu biti DLL ili EXE
– DLL (Dynamic Link Library) je kolekcija klasa i metoda koje se individualno povezuju sa izvršnim programom kada je potrebno (na zahtev)

Assembly sadrži kod u formi MSIL instrukcija i meta informacije u formi metapodataka. (samoopisujući unit)

Manifest – metapodaci o assembly-u

Metapodaci – opisuju tipove i metode u samom kodu

Članice klase sa modifikatorom pristupa **internal**, vidljive su u svim klasama u jednom assembly-a



Imenovane inkapsulacije - druga vrsta inkapsulacija koja se koristi za konstruisanje velikih programa

- Veliki programi definišu veliki broj globalnih imena, pa mora postojati način za organizovanje takvih imena u logičke grupe
- **Imenovane inkapsulacije** su *logičke inkapsulacije* i koriste se za definisanje dosega (namespace-a) za imena

Imenovana inkapsulacija u C++

- **namespaces** imenovana inkapsulacija reprezentuje se preko specifikacije **namespace.** Svaka biblioteka ima sopstveni namespace, a za njihovo kori[‘enje izvan namespace koriste se kvalifikovana imena sa imenom namespace

Primer stack: Sve deklaracije za stack smeštene su u njegov sopstveni namespace blok

```
namespace MyStack {
 //deklaracije
        int top = -1;
        int stack [10];
         ….
}
```

- Implemetaciona datoteka za ATP stack može da referencira deklarisana imenima u namespace bloku preko operatora dosega **::**

  MyStack**::**top
- Umesto operatora dosega, klijent koristi **using** za imena iz drugih namespace-ova

  using namespace MyStack

Imenovana inkapsulacije u programskom jeziku C#

- imenovana inkapsulacija u C# je **namespace**
- Namespace-ovi su programski elementi projektovani prvenstveno za organizovanje programskog koda
- mogu se organizovati u hijerarhijsku srukturu, koja se reprezentuje preko ugnježdavanja namespace-ova
- Članice namespace mogu da budu: klase, strukture, delegati, interfejsi

```
using System; // deklaracije namespace
namespace N1
{
    namespace N2
    {
        class myExample
        {       public static void myPrint*(
                {
                  Console.WriteLine("primer poziva clanice namesp.");
                }
        }
    }
    class NamespaceCalling
    {   public static void Main ()
        {
                N2.myExample.myPrint();
        }
    }
}               // Primer za ugnježdene namespacese-ove u C#
```

Imenovana inkapsulacija u programskom jeziku Java

- Imenovana inkapsulaciona konstrukcija je **package** koji može da sadrži više definicija klasa
- Članice definisane u klasi koja je sadržana u package-u mogu biti **private**, **public**, **protected** ili su bez modifikatora pristupa, pa su u tom slučaju vidljive za sve klase u package (*package scope*)
- Deklaracija paketa: **package** myStack;
- Klijent package-a referencira imena definisana u package-u koristeći ili
  - puno kvalifikovano ime - myStack.top
  - ili deklaraciju **import,** koja dozvoljava skraćene reference za imena def. u package-u

  import myStack.*;

Imenovane inkapsulacije u programskom jeziku Ada

- **package** se koristi kao imenovana inkapsulacije
- Definiše se hijerarhija paketa, koji se memorišu u odgovarajuće hijerarijske direktorijume

10. Uvod u "markup" jezike: XML tehnologije

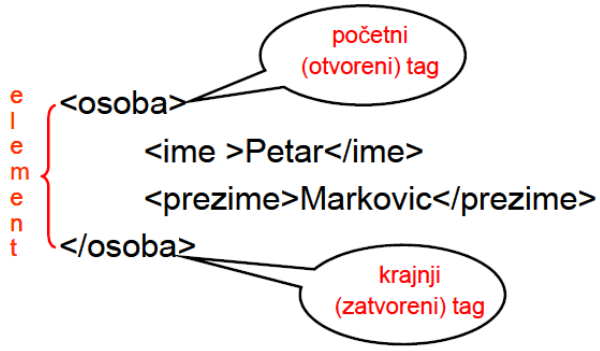
Osnovni koncepti XML-a

XML eXtensible Markup Language
XML je danas postao *de-facto* standard za opis sadržaja i logičke strukture (tekstualnih I multimedijalnih) dokumenata i razmenu dokumenata na Web-u.

• ***Markup***
- Oznake pomoću kojih se dodaju posebna značenja podatku
- U XML koristi se **tag** za predstavljanje markup-a

• ***Extensible***
- *proširljiv* jezik, dozvoljava definisanje novih tag-ova
- meta jezik omogućava definisanje drugih markup jezika

XML dokument se sastoji iz: podataka (teksta) i tagova.



XML dokument: samoopisujuća, platformski nezavisna tekstualna datoteka
- Informacije o podacima sastavni su deo dokumenta sa podacima
- Zbog toga je XML samoopisujući i veoma pogodan za razmenu podataka

Nesamoopisujući modeli (relacioni, objektno-orjentisani):

Data part:
("Students", {["John", s111111111, [123,"Main St"]],
["Joe", s222222222, [321, "Pine St"]] }
])
Schema part:

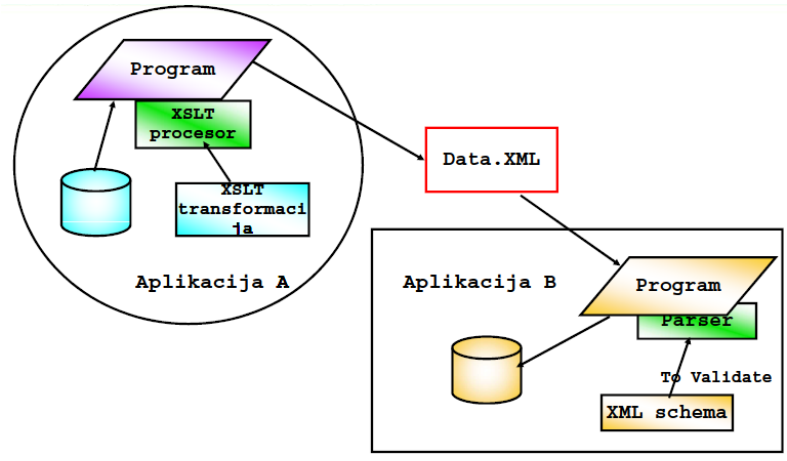
PersonList [*ListName*: String,
Contents: [*Name*: String,
Id: String,
Address: [*Number*: Integer, *Street*: String]]
]

Samoopisujući modeli (na primer XML):
([ListName: “Students”,
Contents: { [Name: “John”,
Id: “s111111111”,
Address: [Number: 123, Street: “Main St.”]] ,
[Name: “Joe”,
Id: “s222222222”,
Address: [Number: 321, Street: “Pine St.”]] }
])

Razdvajanje struktuiranog sadržaja dokumenta od njegove prezentacije
- HTML za formatiranje i prikazivanje dokumenata
- XML za struktuiranje dokumenata

Korišćenje XML-a
XML je infrastruktura za druge tehnologije:
– *Procesiranje XML dokumenata*
- XML parseri
- transfromacija XML dokumenata (XSLT jezik)
– *Specifikacija logičke strukture XML dokumenata*
- DTD = Document Type Definiton
- XML schema
– *Upitni XML jezici*
- XPath
- XQuery

XML je prvenstveno projektovan za distribuirano okruženje
– XML je veoma pogodan kao format za razmenu podataka izmenu heterogenih aplikacija na Web-u
– Samoopisujući
– Tekstualni format (XML format)
– Zajednička sintaksa
– Razvijeni parseri
– XML kao format je dovoljno formalan za mašinsko procesiranje i dovoljno razumljiv za korisnike



Korišćenje XML-a

Web servisi - nov standard za kreiranje interoperabilnih distribuiranih aplikacijasi
Skup standarda zasnovani na XML-u:
- SOAP = Simple Object Access Protocol
- WSDL = Web Service Definition Language
- UDDI = Universal Description, Discovery and Integration Protocol

Korišćenje XML-a

Memorisanje XML podataka:
– XML datoteke i XML baze
– Realcione baze
- Transformacija XML dokumenta u relacione tabele
- XML dokumenta memorišu se u kolone (čiji je tip - XML type) relacionih tabela

Istrorijski razvoj XML-a

W3C = World Wide Web Consortium (organizacija za standardizaciju Web tehnologija)
1996. W3C počela razvoj standarda za XML sa motivacijom da XML treba da kombinuje
Fleksibilnost **SGML**
SGML **= S**tandard **G**eneralized **M**arkup **L**anguage
jednostavnost **HTML**
HTML = **H**yper**T**ext **M**arkup **L**anguage
U februaru 1998. definisan je **XML 1.0 standard**

SGML

standard za definisanje i reprezentovanje structure različitih tipova elektronskih dokumenta, (ISO standard 1985)
- tagovi se koriste samo za označavanje strukture dokumenta
- proširljiv jezik, dozvoljava definisanje novih tagova
- meta jezik standard za definisanje novih markup jezika
- Veoma složen jezik

XML je podskup SGML (prilagodjen potrebama Web-a)

HTML

Početkom 1990. HTML je definisan od W3C kao standard
- definisan u SGML
- koristi *fiksni* broj *predefinisanih* tagova
- prvenstveno projektovan za formatiranje I prezentovanje dokumenta na Web-u

Primeri	*Prikaz u browser-u*
bold	**bold**
italic	*italic*

XML dokumenta

XML dokumenta su samoopisujuće, platformski nezavisne tekstualne datoteke
XML dokument sadrži :
- Tekst (sadržaj dokumenta)
- tag-ove

<pozdrav> Hello XML!</pozdrav>

Početni (otvoren) tag — tekst — Krajnji (zatvoren) tag

Elementi su osnovni blokovi XML-a

Kontejner element par tag-ova (početni i krajnji tag) sa sadržajem

<pozdrav> Hello XML! </pozdrav>

Prazan element obično se za krajnji tag koristi skraćenica **/>**

<poruka/>

<pozdrav tekst = "Hello XML" />

Elementima se mogu pridružiti **atributi**
XML atributi obezbenuju dodatne informacije o elementima

naziv atributa *vrednost atributa*

```
<poruka datum = "12.5.08." sala = "201" >
        <tekst> Sastanak Katedre sutra u 10 </tekst>
</poruka>
```

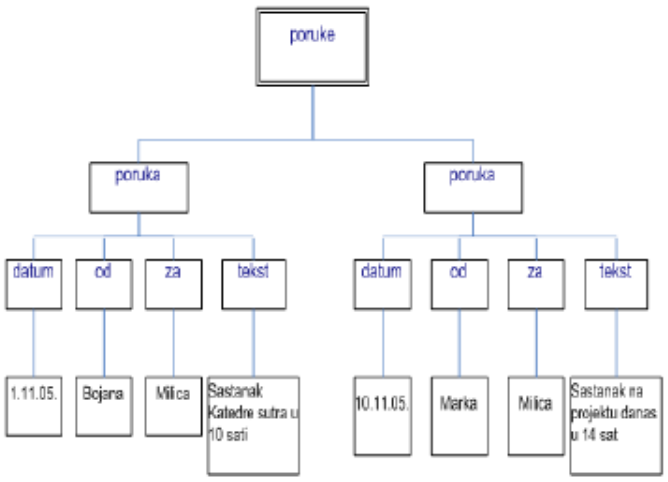
Imena XML tagovi i imena atributa – *case sensitive*

Struktura XML dokumenta

- hijerarhijska struktura (stablo) koja se sastoji iz elemenata, atributa i znakovnih podataka
- XML dokument ima jedan i samo jedan koreni (root) element
- Svi ostali elementi u strukturi su elementi "deca" korenog element (dozvoljeno višestruko ugnježdavanje elemenata)

Hijerarhijska struktura XML dokumenta

```
<?xml version="1.0" encoding="UTF-8"?>
<poruke>
  <poruka>
    <datum>1.11.05. </datum>
      <od>Bojana</od>
       <za>Milicu</za>
       <tekst>Sastanak Katedre sutra u 10 sati </tekst>
  </poruka>
  <poruka>
    <datum>10.11.05. </datum>
      <od>Marka</od>
       <za>Milicu</za>
       <tekst>Sastanak na projektu danas u 14 sati  </tekst>
  </poruka>
</poruke>
```



XML deklaracija

Svaki XML dokument mora da sadrži XML deklaraciju, tj. ***instrukciju obrade*** kojom se dokument identifikuje kao XML dokument.

- Osnovni oblik XML deklaracije: <?xml version =“1.0”?>
- Opcioni oblik XML deklaracije: <?xml version =“1.0” encoding= “UTF-8”?>

<?xml version=“1.0” encoding=“UTF-8”?>

- **?** oznaka za instrukciju obrade - instrukcija obrade je poruka programima koji procesiraju XML dokument
- atribut version specificira XML verziju
- atribut encoding definiše znakovni kod u kome je XML document napisan
- UTF-8 (kompresovana verzija Unicode-a)
- UTF-16 (Unicode)

UTF Unicode Transformation Format

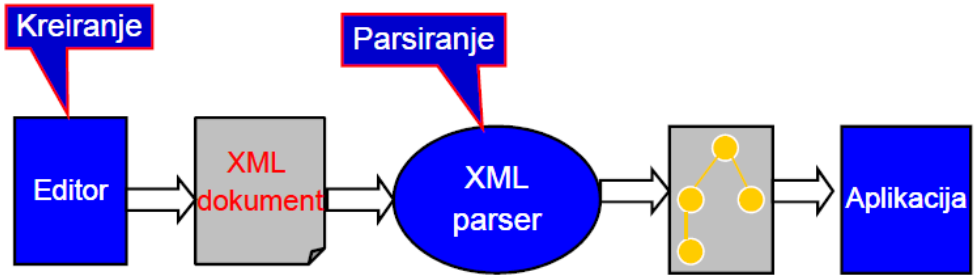
Dobro-oformljen XML document

- postoji XML deklaracija
- dokument sadrži jedan i samo jedan koreni element u kome su ugnježđeni svi ostali elementi i njihovi sadržaji
- svi elementi i atributi u dokumentu moraju da budu sintaksno ispravni

Provera sintaksne korektnosti XML dokumenta

- XML parser verifikuje da li je XML dokument dobrooformljen
- XML parser čita dokument i konvertuje ga u hijerahijsku strukturu
- XML parser prenosi parsirani dokument do krajnje Aplikacije

Obrada XML dokumenta



Kreiranje XML dokumenta

- Tekst editori (na primer Notepad)
- VS.NET XML Desinger
- XML Spy – razvojno okruženje za XML

Pregled XML dokumenata (source)

Pomoću web browser-a koji podržavaju XML (Internet Explorer 5.0 i više verzije)

Validni XML document

- Dobro-oformljen
- Konzistentan sa strukturom definisanom u opisu tipa dokumenta

Definisanje tipova XML dokumenata

W3C je ponudio dva standarda načina za definisanje tipova XML dokumenta, odnosno opisivanje strukture XML dokumenta:

- **Document Type Definiton (DTD)**
- **XML Schema Definition (XSD)**

Definisanje tipova XML dokumenata
DTD i XSD definišu:
– ***strukturu*** XML dokumenta
– ***ime*** i ***tip*** svakog XML elementa/atributa
(DTD- ograničene mogućnosti za definisanje tipova)

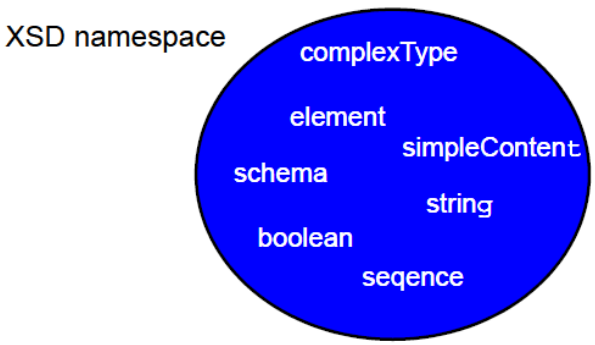
DTD
– Nasledjen od SGML-a
– Poseban jezik za opis strukture dokumenta
– Vrlo ograničene mogućnosti za definisanje tipova

Primer opisa strukture dokumenta pomoću DTD
<!ELEMENT Knjige (Knjiga+)>
<!ELEMENT Knjiga (Naslov, Autor, Godina, ISBN, Izdavac)>
<!ELEMENT Naslov (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
<!ELEMENT Godina (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Izdavac (#PCDATA)>
– #PCDATA – Parser Character Data, oznacava znakovni sadržaj
– "+" – element se pojavljuje bar jednom

XML Schema
– preporuka W3C od maja 2001
– XML schema je data preko XML sintakse (XML šema je XML dokument)
– podržava definicije prostih i složenih tipova I poseduje napredne mehanizme za grupisanje XML elemenata u XML dokumentu

Za konstrukciju svake XML šeme koriste se:
– **Schema** element (koreni element svake XML šeme)
– Deklaracije elemenata
– Deklaracije atributa
– Definicije prostih i složenih tipova

Schema element ukazuje na *definiciju* XML šeme u kojoj se nalaze svi potrebni elementi za kreiranje XML šeme



Deklaracija XSD namespace:
xmlns:prefix="namespace name"
- prefix se koristi kao skraćeno ime za "namespace name" u XML šemi
- "namespace name" je lokacija definicije XSD i specificira se preko URL

Primer

<?xml version="1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
……
</xsd:schema>

Deklaracija elemanata u XML šemi

• Za svaki *element* u XML šemi definiše se **naziv** i **tip** (atributi *name* i *type*)

<xsd:element name =“Autor” type = “xsd:string”/>

• Tip može da bude:

- korisnički definisan tip (pr. ComplexType)
- ili je u opsegu imena definicije XML šeme (primer string)

• Kardinalnost elementa može se specificirati u njegovom ocuelementu;

inače po difoltu, kardinalnost elementa je:

minOccurs = “1”

maxOccurs = “1”

Definisanje složenih tipova

Složeni tipovi se konstruišu od prostih i drugih složenih tipova korišćenjem konstruktora:

– **sequence** def. urenenu grupu elemenata.

Po difoltu, svaki element je obavezan i jednoznačan

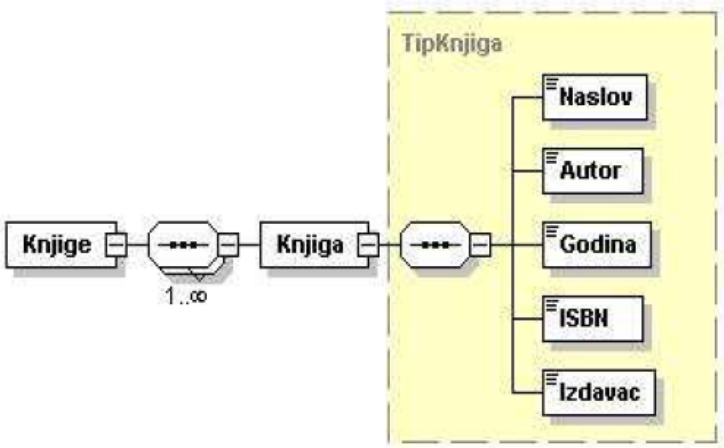
– **choice** def. grupu iz kojih se mogu izvlačiti pojedinačni elementi

– **all** def. grupu u kojoj se svi elementi mogu pojaviti maksimalno jedanput.

Primer XML šeme za tip Knjiga:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
   <xsd:element name="Knjige">
      <xsd:complexType>
         <xsd:sequence maxOccurs="unbounded">
            <xsd:element name="Knjiga" type="TipKnjiga"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="TipKnjiga">
      <xsd:sequence>
         <xsd:element name="Naslov" type="xsd:string"/>
         <xsd:element name="Autor" type="xsd:string"/>
         <xsd:element name="Godina" type="xsd:string"/>
         <xsd:element name="ISBN" type="xsd:string"/>
         <xsd:element name="Izdavac" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Prikaz XML šeme preko strukture stable



XML dokument formiran *u skladu* sa XML šemom knjige.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<Knjige xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Milica\FolderXML\knjige.xsd">
   <Knjiga>
      <Naslov>Baze podataka</Naslov>
      <Autor>Branislav Lazarevic</Autor>
      <Godina>2003</Godina>
      <ISBN>86-80239-96-8</ISBN>
      <Izdavac>FON, Beograd</Izdavac>
   </Knjiga>
   <Knjiga>
      <Naslov>Principles of Database and Knowledge-Base Systems</Naslov>
      <Autor>Jeffrey Ulman</Autor>
      <Godina>1998</Godina>
      <ISBN>0-7167-8158-1</ISBN>
      <Izdavac>Freeman</Izdavac>
   </Knjiga>
</Knjige>
```

11. Procesiranje XML dokumenata

Modeli XML parsera

XML parser je softver koji čita XML dokument i čini dostupnim njegov sadržaj i strukturu aplikaciji preko API-a
API = Application Programming Interfaces

Postoje dve vrste parsera:
1) XML parseri koji verifikuju samo sintaksnu ispravnost XML dokumenta (da li je XML dokument dobro oformljen)
2) XML parseri koji vrše validaciju XML dokumenta u skladu sa XML šemom ili DTD

Dva osnovna **modela** XML parsera:
SAX model = **S**imple **A**PI for **X**ML
DOM model = **D**ocument **O**bject **M**odel

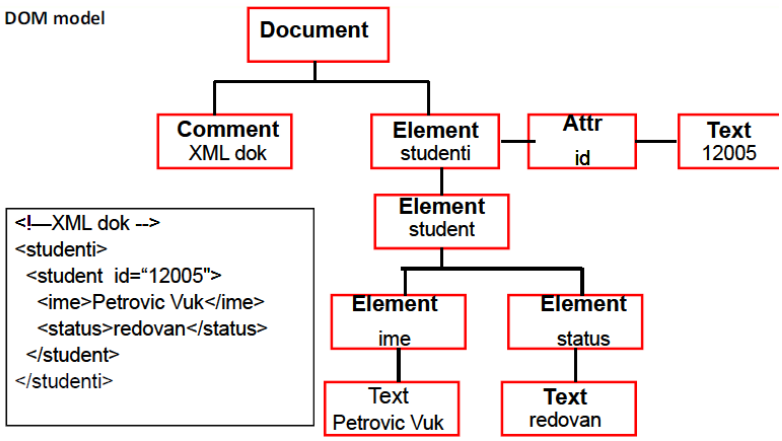
Koraci obrade XML dokumenata:
1) Parsiranje XML dokumenta (korišćenjem XML parsera)
   - Parser formira memorijsko stablo čvorova (DOM),
   - Parser, za vreme parsiranja, šalje događaje aplikaciji (SAX)
2) Obrada dokumenta
   - Aplikacija pristupa i menja čvorove stabla korišćenjem interfejsa
   - DOM API
   - Aplikacija obranuje poslate SAX doganaje
3) Interpretacija parsiranog XML dokumenta u aplikaciji

DOM model (Document Object Model)
• Standardni objektno-orjentisani programski interfejs za obradu XML dokumenata
• W3C standard
• W3C DOM specifikacija pruža samo definiciju interfejsa za DOM biblioteke, a ne i detalje njihove implementacije

DOM model reprezentuje XML dokument kao memorijsko stablo čvorova (DOM stablo) i omogućava, (preko skupa svojih interfejsa), *navigaciju* i *izmene* dokumenta
Preko ovog modela se iz softverskih aplikacija može manipulisati sa XML dokumentima i njihovim elementima kao sa objektima

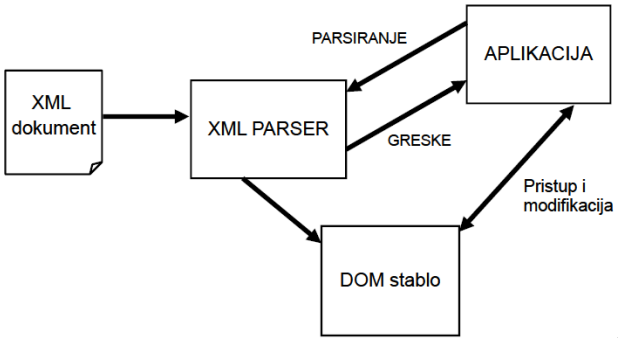
Tipovi čvorova u W3C specifikaciji:
- Document
- Element
- Attribute
- Character data
- Text
- Comment

Metode za pristup i modifikaciju čvorova DOM stable



Obrada XML dokumenta korišćenjem DOM parsera
- čita XML dokument od početka do kraja
- formira u memoriji strukturu stabla (DOM stablo) koja reprezentuje strukturu i sadržaj takvog dokumenta

Proces obrade XML dokumenata korišćenjem DOM parsera



DOM model
Prednosti
– Dinamički pristup i modifikacija čvorova DOM stabla
– Efikasno pretraživnje koje se zasniva na strukturi stabla
– Isti interfejs za različte programske jezike (C++, Java, C#, ...)
Nedostaci
– Može da bude spor i zahteva dosta memorijskih resursa

Korišćenje DOM parsera
– Kada se zahteva obrada većine elemenata u XML dokumentu
– Kada se zahteva dinamički pristup i manipulacija sa XML dokumentom i njegovim elementima
– XML dokumenta sa složenom strukturom

Implementacija DOM parsera

DOM parseri MS

XmlDocument (.NET Framework)

MSXML (Microsoft XML Parser ili Microsoft XML Core Services) je COM (Component Object Model) implementacija W3C DOM modela. Različite verzije MSXML su uključene u različite MS proizvode kao što su MS Internet Explorer, MS Office, MS SQL Server. MSXML obezbenuje još i sledeće servise: XSD, XSLT 1.0, SAX, XPath

Java

– JAXP (Java API for XML Processing) obezbenuje sledeće servise:

- DOM
- SAX
- XSLT

SAX model (Simple API for XML)

• *Event-based* model (zasnovan na doganajima)
• Razvijen od strane XML-DEV grupe
• Industrijski standard
Verzija 1.0 1998
Verzija 2.0 2000

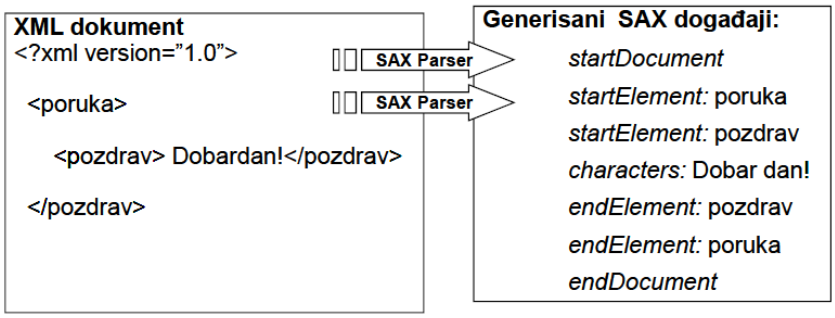
Obrada XML dokumenta korišćenjem SAX parsera

SAX parser čita XML dokument kao stream XML tagova

– Prilikom čitanja XML dokumenta, SAX parser generiše dogadjaje kad god otkrije element/atribut/tekst/instrukciju obrade i šalje aplikaciji

Aplikacija obranuje doganaje generisane od strane parsera

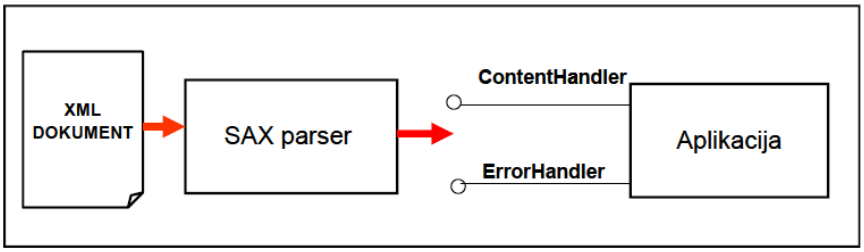
– aplikacija implementira odgovarajuće hendlere koji sadrže metode kojima se ti dogadjaji obranuju



– sekvencijalan i

– "forward only" pristup (svaki element se parsira naniže sve do listova pre nego što se prene na sledeći element istog nivoa)

SAX – "push" model XML parsera



ContentHandler interfejs definiše metode koje se pozivaju za procesiranje generisanih događaja (*startDocument(), endDocument()*, *startElement()*, ...)

SAX parser

• *Prednosti*
  – Efikasan (veoma brz, ušteda memorije)
• *Nedostaci*
  – ne kreira memorijsko stablo za reprezentovanje XML dokumenta
  – sekvenicjalni pristup komponentama dokumenta

Korišćenje XML parsera

• Kada se ne zahteva dinamički pristup i izmena elemenata XML dokumenta
• Obrada dokumenta sa prostom strukturom koja sadrže veliki obim podataka

SAX parseri:

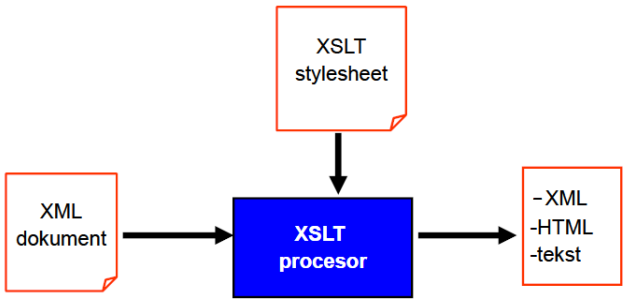
- **Xerces** (http://xml.apache.org)
- **Oracle XML Parser**
- **Project X** (Sun)
- **XML4J** (IBM)
- **MSXML 4.0 (** Microsoft)

XSLT processor

XSLT eXtensible Stylesheet Language Transformation

• XSLT je deklarativni jezik koji se koristi za opis pravila transformacija XML dokumenta u
  - drugi XML dokument
  - HTML dokument
  - tekst
• W3C standard
• XSLT verzije:
  - XSLT 1.0 (Novembar 1999)
  - XSLT 2.0 (Novembar 2002)

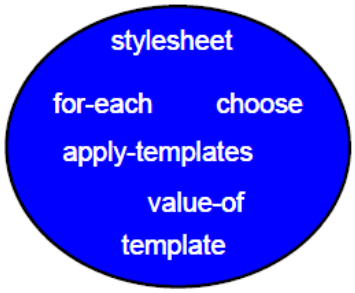
XSLT transformacioni process



XSLT stylesheet dokument je XML dokument

- XSLT instrukcije se izražavaju kao XML elementi

Elementi koji se koriste za konstruisanje stylesheet dokumenta definisani su preko XSLT namespace-a, lokacija se spec. preko URL-a: http://www.w3.org/1999/XSL/Transform



Osnovne karakteristike XSLT stylesheet dokumenta
stylesheet element je koreni element
<?xml version="1.0"?>
<xsl:stylesheet version "1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
.....
.....
</xsl:stylesheet>

Stylesheet element sadrži skup templejt pravila koja se deklarišu sa **<xsl:template>** elementima
pravila opisuju kako se pojedini elementi u XML dokumentu transformišu u rezultujuće elemente u izlaznom dokumentu

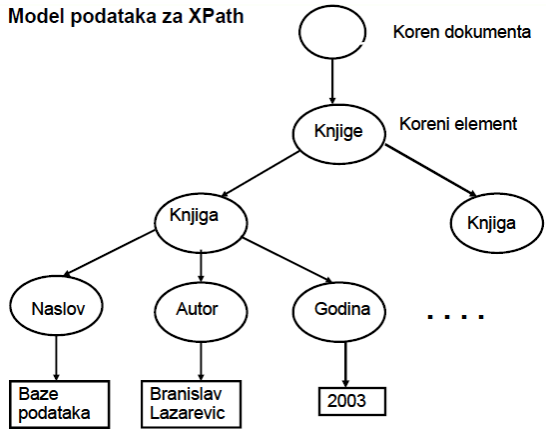
Templejt pravilo sadrži dva dela:
- ***Pattern*** identifikuje i izdvaja elemente *ulaznog* XML dokumenta na koje će biti primenjena transformacija
- ***akcija*** opisuje transformaciju koja se primenjuje

<xsl:template match="*pattern*">
[akcija]
</xsl:template>

Templejt pravila
- match atribit koristi se za povezivanje templejta sa nekim delom ulaznog XML dokumenta
- Vrednost match atributa je XPath pattern

<xsl:template match="*XPath pattern*">
[akcija]
</xsl:template>

XPath (*XML Path Language)*
• **XPath** je jezik koji omogućava navigaciju do delova(čvorova) XML dokumenta (kao što su elementi, atributi, njihove vrednosti)
• XPath je W3C standard
• U XSLT dokumentu XPath se koristi za izdvajanje delova XML dokumenta na koje će biti primenjena transformacija
• XML dokument se ovde posmatra kao stablo čvorova sa definisanim čvorom koji se naziva ***koren dokumenta***
- koren dokumenta je bezimeni čvor čije je dete *koreni element* XML Dokumenta



Xpath
Čvor se adresira preko tzv. *izraza putanje* – niz od jednog ili više lokacijskih koraka

Primer
/Knjige/Knjiga/Godina
Rezultat: <Godina>2003</Godina>

<Godina>1998</Godina>
Izraz putanje koji pocinje sa "/" reprezentuje apsolutnu putanju **"/"** XPath pattern za koren dokumenta

<xsl:template match="/">
…….
</xsl:template>
atribut **match="/"** povezuje templejt sa korenom dokumenta

Telo templejt pravila satoji se iz:
- XSLT instrukcija
- Elemenata koji specificiraju neki željeni izlazni tekst koji XSLT procesor treba da ubaci u izlazni document

Primer XSLT instrukcije *value-of*
<xsl:template match="/">
<xsl:value-of select = "pozdrav"/>
</xsl:template>
- sadržaj elementa **"pozdrav"** prvo se dodeljuje atributu *select*
- zatim, sadržaj elementa **"pozdrav"** XSLT procesor kopira u izlazni Document

Primer elemenata za specifikaciju izlaznog teksta
Pretpostavka: izlazni dokument je HTML dokument

<xsl:attribute name="**color**">**blue**</xsl:attribute>
<xsl:attribute name="**size**">**6**</xsl:attribute>
pozdrav xml programera

Primer elemenata za specifikaciju izlaznog teksta

XML dokument (pozdrav.xml):

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
        href="pozdrav.xsl"?>
<Pozdrav>
        Hello XML!
</Pozdrav>
```

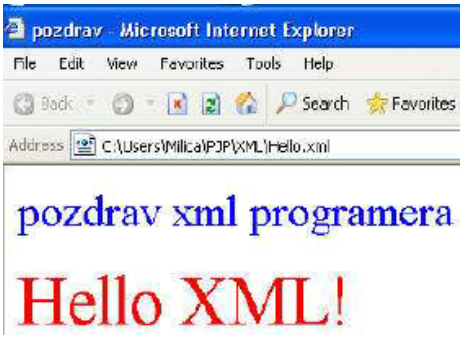
*Željeni izlaz- **HTML** dokument*

```
<html>
  <head>
        <title>pozdrav</title>
  </head>
  <body>
    <p>
        <font color="red"
        size="14">
            pozdrav xml programera
        </font>
    </p>
    <p>
            <font color="blue"
                size="16">
          Hello XML! </font>
    </p>
  </body>
</html>
```

XSLT Stylesheet dokument (pozdrav.xsl)

```
<?xml version="1.0" ?>
  <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html>
     <head>
        <title>pozdrav</title>
     </head>
    <body>
       <p> <font>
              <xsl:attribute name="color">blue</xsl:attribute>
              <xsl:attribute name="size">6</xsl:attribute>
              pozdrav xml programera
       </font> </p>
       <p> <font>
          <xsl:attribute name="color">red</xsl:attribute>
                <xsl:attribute name="size">16</xsl:attribute>
          <xsl:value-of select="Pozdrav" />
       </font></p>
    </body>
  </html>
</xsl:template>
 </xsl:stylesheet>
```

Rezultat transformacije



XSLT procesori:

- Standalone XSLT procesori
  Java XSLT procesor, SAXON, Oracle XSLT, Xalan (Apache projekat)
- Korišćenje Web Browser-a za XSLT transformacije
  - MS Internet Explorer 5.5 i više verzije
    XSLT procesor u IE je deo MSXML parsera
  - Netscape 6.0
  - JavaScript
- Korišćenje Web servera za XSLT transformacije
  Tri načina za izvršavanje XSLT transformacija
  - Java servleti
  - ASP (Active Server Pages)
  - JSP (Java Server Pages)

12. MS implementacija W3C XML standarda

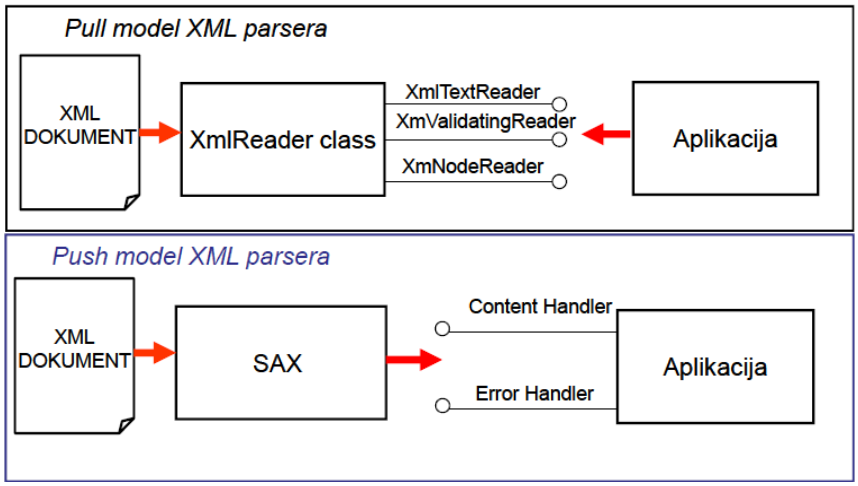
Implementacija W3C XML standarda u .NET Framework-u

W3C XML Standard	.NET Framework namespace	.NET XML klase
XML 1.0	System.Xml	
XML Schema	System.Xml System.Xml.Schema	XmlSchema
XSLT	System.Xml.Xsl	XslTransform
XPath	System.Xml System.Xml.Path	Path
DOM	System.Xml	XmlDocument

Obrada XML dokumenata u .NET Framework-u

Dva načina obrade XML dokumenata:

- Preko DOM modela korišćenjem **XmlDocument** klase
- Preko Pull modela korišćenjem **XmlReader** klase. Pull model je nov pristup za rad sa XML dokumentima

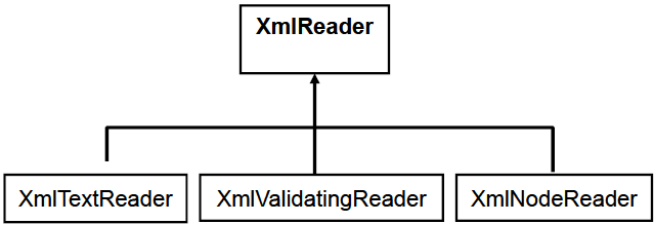


Pull&Push modeli parsera

• Pull model ne formira za XML dokument memorijsko stablo (slično kao i SAX)
• SAX je push model – dostavlja doganaje aplikaciji koja ih obranuje
• Pull model omogućava da aplikacija zahteva prolaženje kroz XML dokument i zatim selektovanje i pristup samo zahtevanim čvorovima
• Prednost Pull modela – Poboljšava performanse XmlReader-a

XmlReader klasa

• XmlReader je apstraktna klasa
• Reprezentuje pull model XML parsera
• Memorijski efikasan, forward-only, read-only pristup XML podacima



XmlTextReader klasa

• Provera da li je XML dokument dobro-oformljen
  – Ne proverava validnost
• Konstruktori omogućavaju čitanje XML iz različitih ulaznih izvora
  - datoteka, stream objekat ili TextReader
• *Read()* metoda omogućava navigaciju kroz čvorove XML dokumenta
  - obezbenuje načine za čitanje sadržaja dokumenata, elemenata i atributa

XmlTextReader klasa

Korišćenje XmlTextReader-a:

• Istanciranje XmlTextReader objekta
• Čitanje i obrada podataka
  - Parsira XML dokument korišćenjem *Read()* metode u *While* petlji

XmlTextReader klasa

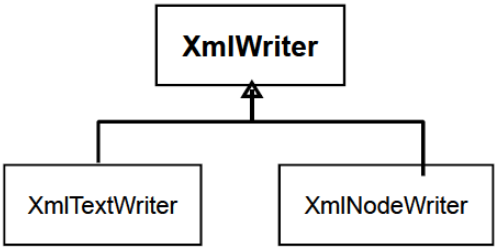
```
// konstruktor
XmlTextReader reader = new XmlTextReader ("Studenti.xml");
while (reader.Read())
{
// obrada tekućeg čvora
        switch (reader.NodeType)
        {
                case XmlNodeType.Element :
                        Console.Write("<" + reader.Name +  ">");
                        break;
                case XmlNodeType.EndElement:
                        Console.Write("</" +  reader.Name + ">");
                        break;
                case XmlNodeType.Text :
                        Console.Write(reader.Value);
                        break;
                default:Console.WriteLine();break;
        }
}
reader.Close ();
```

XmlValidatingReader& XmlNodeReader klase

- **XmlValidatingReader** obezbenuje podršku za *validaciju* XML dokumenta u skladu sa XSD
- **XmlNodeReader** omogućava čitanje podstabla XML DOM stabla
  - Ne podržava validaciju

xmlWriter klasa

- Programsko generisanje XML dokumenata u datoteku, stream, TextWriter
- XMLWriter je apstraktna klasa
- Reprezentuje brzi, forward-only, memorijski efikasan XML writer



xmlTextWriter klasa

Korišćenje XmlTextWriter-a:

1) Istanciranje XmlTextWriter-a (konstruktor)
2) Postavljanje property-a (za formatiranje itd.)
3) Izvršavanje *Write* metode za generisanje XML
4) Izvršavanje *Close()* metode

```
//konstruktor
XmlTextWriter writer = new XmlTextWriter
("knjiga.xml",System.Text .Encoding .UTF8 );
//kreiranje root elementa
writer.WriteStartElement ("Knjiga");
//kreiranje elementa Naslov
writer.WriteStartElement ("Naslov");
writer.WriteString ("Profesional ASP.NET");
writer.WriteEndElement();
//kreiranje elementa Autor
writer.WriteStartElement ("Autor");
writer.WriteString ("Richard Anderson");
writer.WriteEndElement();
//kreiranje elementa ISBN
writer.WriteStartElement("ISBN");
writer.WriteString ("86-7991-155-0");
writer.WriteEndElement();
//kreiranje elementa Datum
writer.WriteStartElement("Datum");
writer.WriteString ("Jun 2002");
writer.WriteEndElement();
//kreiranje elementa Izdavac
writer.WriteStartElement("Izdavac");
writer.WriteString ("Wrox Press, Birmingham");
writer.WriteEndElement();
//zatvaranje root elementa
writer.WriteEndElement ();
writer.Flush ();
writer.Close ();
```

XMLDocument klasa

• XmlDocument obezbedjuje podršku W3C DOM modela
  - Reprezentuje XML dokument kao memorijsku strukturu stable
  - Metode: Load(), Save()
• Izvedena iz XmlNode klase

• Korišćenje **XmlDocument** klase

```
// konstruktor
XmlDocument doc = new XmlDocument ();
// ucitavanje XML dokumenta
doc.Load ("Studenti.xml");
// ispis sadržaj xml dokumenta
Console.WriteLine(doc.InnerXml.ToString ());
```

.NET Klase: XslTransform klasa

• System.Xml.Xsl namespace
• transformiše ulazni XML document korišćenjem XSLT stylesheet-a
• Ključne metode
- *Load*
- *Transform*

```
// kreiranje XslTransform objekta
XslTransfrom transformacija = new XslTransform;
// punjenje stylesheet doc
transformacija.Load("pozdrav.xsl");
// transformacija
transformacija.Transform("pozdrav.xml",
"pozdrav.html");
```

```
string XmlPath = "Hello.xml";
string ResultHtmlPath = "Hello.html";
XmlTextReader xslt = new
XmlTextReader("Hello.xst");
XslTransform transformacija = new
XslTransform();
transformacija.Load(xslt);
transformacija.Transform(XmlPath,
ResultHtmlPath);
```

13. XML Parseri

XML parser, je u suštini, softver koji čita XML datoteku i čini dostupnim podatke koji se u njoj nalaze.

Postoje dva različita pristupa implementaciji XML parsera:

Parser zasnovan na dogadjajima – obradjuje XML dokument sekvencijalno, tretirajući komponentu po komponentu (**SAX** pristup).

Parser zasnovan na stablu – predstavlja ceo dokument u obliku stabla i obezbedjuje pristup pojedinačnim čvorovima stabla (**DOM** pristup).

Implementacija XML parsera

DOM i SAX su nezavisni API-ji koji su podržani u različitim jezicima.
W3C DOM specifikacija pruža samo definiciju interfejsa za DOM biblioteke, a ne i detalje njihove implementacije.
SAX nije W3C preporuka već industrijski standard.
Ne mogu se koristiti za parsiranje dokumenata već je potreban parser koji implementira interfejse definisane datim specifikacijama.
Parser je konkretna implementacija SAX-a ili DOM-a razvijena za odgovarajuću softversku i hardversku platformu.

Razvijen je čitav niz XML parsera za različite softverske i hardverske platforme koji podržavaju i SAX i DOM pristup.

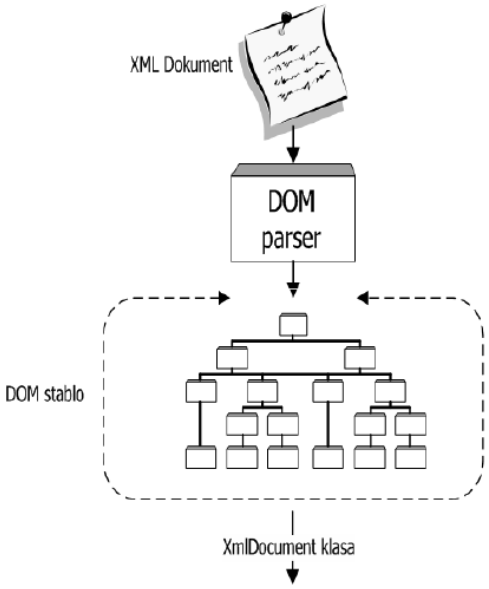
- Microsoft – MSXML
- Java - Xerces2, XML4J, Project X, JDOM
- C – Expat

Microsoft-ova tehnologija za parsiranje XML dokumenata
Microsoft je razvio XML parser tehnologiju koja omogućava programski pristup XML podacima.
Ovaj parser se naziva **Microsoft XML Parser** (kada su u pitanju verzija 3.0 i prethodne verzije) odnosno, **Microsoft XML Core Services** (za verziju 4.0).
Za oba se koristi skraćenica **MSXML** _ MSXML omogućava korisnicima da razviju aplikacije zasnovane na XML-u.

Medju osnovnim servisima koje MSXML pruža nalazi se i podrška projektantima za:
Document Object Model (DOM), standardna biblioteka interfejsa aplikacionih programa (API) za pristup XML dokumentima.
Simple API for XML (SAX2), programska alternative obradi zasnovanoj na DOM-u.

XML parseri u .NET okruženju
.NET Framework pruža dva načina parsiranja XML dokumenata:
Pull-model parser (**XmlReader** klasa i klase koje su sa njom povezane).
DOM-model parser (**XmlDocument** klasa i klase koje su sa njom povezane).
SAX nije implementiran u .NET Framework-u, iako se može formirati korišćenjem **XmlReader** klase.

DOM parser
Document Object Model (DOM) je objektni model (stablo), kojim se prikazuje struktura I sadržaj XML dokumenta.
Ovim modelom upravlja MSXML parser.



Kada MSXML parser učita XML dokument u DOM, on prvo čita dati dokument, parsirajući njegov sadržaj na taj način što ga razbija na pojedinačne delove:
- elemente,
- atribute,
- komentare…

Zatim kreira (u memoriji) konceptualnu reprezentaciju tog dokumenta u vidu stable čvorova.
Č**vorovima se dakle predstavlja struktura I sadržaj dokumenta.**

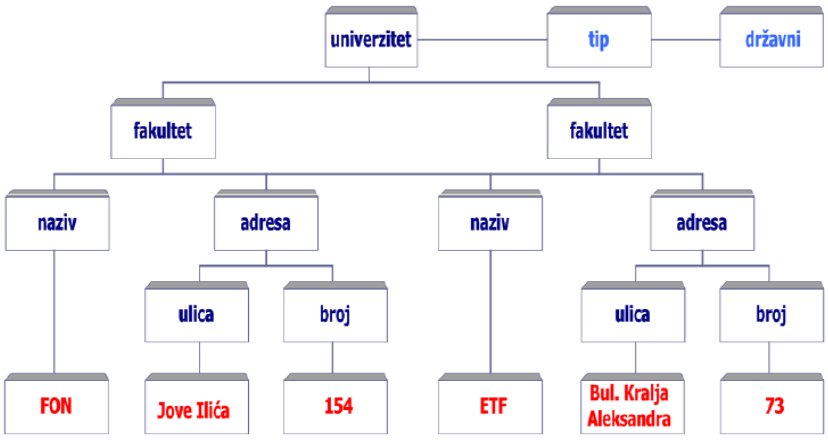
Prilikom kreiranja, svakom čvoru pridružuje se i tip u skladu sa W3C specifikacijom tipova čvorova u DOM-u.
Tip č**vora** odredjuje karakteristike i funkcionalnost datog čvora.
Za većinu dokumenata najčešći tipovi čvorova su:
- element,
- atribut i
- tekst.

Dakle čvorovi zadržavaju i dodatne metapodatke o sebi kao što su njihovi tipovi i vrednosti.
Kada se završi parsiranje dokumenta čvorovi se mogu pretraživati u bilo kom smeru tj. nisu ograničeni na sekvencijalnu obradu.

PRIMER: Univerzitet.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<univerzitet tip="državni">
    <fakultet>
        <naziv> FON </naziv>
        <adresa>
            <ulica> Jove Ilića </ulica>
            <broj> 154 </broj>
        </adresa>
    </fakultet>
    <fakultet>
        <naziv> ETF </naziv>
        <adresa>
            <ulica> Bul. Kralja Aleksandra </ulica>
            <broj> 73 </broj>
        </adresa>
    </fakultet>
</univerzitet>
```

REZULTAT:



DOM tretira XML podatke kao standardni skup objekata.
System.Xml namespace pruža programsku reprezentaciju XML dokumenata i čvorova.
– zasnovan je na W3C DOM Level 1 Core i DOM Level 2 Core preporukama.

Koncepti koje pruža System.Xml namespace:
– **XmlDocument**
– **XmlNode**
– **XmlNodeList**
– **XmlAttribute**
– **XmlAttributeCollection**
– **ChildNodes …**

Dakle, kao rezultat parsiranja dobija se XmlDocument klasa i klase koje su sa njom povezane.

XmlNode objekat je osnovni objekat DOM stabla i predstavlja konkretan čvor (tag).
Objekat klase XmlNode može predstavljati:
– ceo xml document – npr. dokument Univerzitet.xml
– samo jedan čvor – **<naziv>FON</naziv>**

– gde se čvor može sastojati i od drugih čvorova:

```
<fakultet>
    <naziv> FON </naziv>
    <adresa>
        <ulica> Jove Ilića </ulica>
        <broj> 154 </broj>
    </adresa>
</fakultet>
```

– a takone i XML atribut – <univerzitet tip="državni"> gde je tip atribut čvora univerzitet.

Izvedene klase:
– XmlDocument
– XmlDocumentFragment
– XmlAttribute
– XmlEntity
– XmlLinkedNode
– XmlNotation

XmlNode klasa pruža skup metoda i svojstava pomoću kojih se može uzeti ceo čvor, kao i informacije koje taj čvor sadrži (npr. tekst).

XmlNode - polja
Attributes {get;} vraća sve atribute koji pripadaju tekućem čvoru. Atributi su smešteni u klasu - kolekciju: XmlAttributeCollection.
ChildNodes {get;} vraća sve čvorove koji se nalaze unutar tekućeg čvora. Čvorovi su smešteni u klasu – kolekciju: XmlNodeList.
NodeType {get;} enumeracija tipa XmlNodeType koja ukazuje na to kog je tipa čvor u XML dokumentu.
– (*Attribute , Document , Element, Text, XmlDeclaration ....*)
InnerText vraća string koji predstavlja tekst koji se nalazi unutar tekućeg čvora.
InnerXml vraća string koji predstavlja XML čvorove koji se nalaze unutar tekućeg čvora.
OuterXml vraća string koji predstavlja tekući čvor sa svim svojim podčvorovima kao XML element.

XmlNode - metode
AppendChild(XmlNode *node*)
– u tekući čvor ubacuje novi kao poslednji.
PrependChild(XmlNode *node*)
– u tekući čvor ubacuje novi kao prvi.
InsertAfter(XmlNode *newNode*, XmlNode *refNode*)
InsertBefore(XmlNode *newNode,* XmlNode *refNode*)
RemoveChild(XmlNode *oldChild*)
RemoveAll()

INDEXER:
[string name] vraća čvor koji se nalazi unutar tekućeg čvora sa nazivom "name" kao objekat klase XmlNode.

XmlDocument objekat predstavlja konkretan XML dokument I sadrži sve informacije o njemu.
Sam dokument se posmatra kao jedan čvor koji predstavlja vrh stabla i sadrži sve ostale čvorove uključujući i čvor koji predstavlja koreni element dokumenta, koji dalje sadrži sve čvorove u dokumentu: elemente, atribute.

POLJA (SVOJSTVA):
• **DocumentElement {get;}**
  – vraća root element XML dokumenta.
• **XmlDocument** klasa obezbedjuje metode za manipulaciju dokumentom u celini kao što su:
  – učitavanje postojećeg dokumenta u memoriju,
  – čuvanje XML-a u datoteku
  – pregled i manipulaciju svih čvorova u dokumentu.

• Nisu prikazana ona polja i metode koje se naslenuju od klase XmlNode.

XmlDocument – metode
CreateAttribute(string *name*)
– pravi objekat tipa XmlAttribute sa nazivom name.
CreateElement(string *name*)
– pravi objekat tipa XmlElement sa nazivom name.
– Klasa XmlElement je izvedena iz klase XmlLinkedNode, koja je izvedena iz klase XmlNode.
GetElementsByTagName(string *name*)
– vraća sve tagove koji se zovu "name" a pripadaju tekućem čvoru.
Load(string *filename*)
– učitava XML dokument sa lokacije *filename*.
Save(string *filename*)
– snima promene nastale u memoriji u XML dokument na lokaciji *filename.*

Kada postoji referenca na dokument moguće je kretanje po hijerarhiji.
Preko **XmlDocument** objekta može se pristupiti **ChildNodes** svojstvu, koji pruža mogućnost pristupa svim čvorovima u dokumentu, sa vrha na dole.
– Čvorovi se pojavljuju u istom redosledu u kome su se nalazili u XML datoteci.
ChildNodes svojstvo imaju i svi ostali čvorovi u hijerarhiji.

Predstavlja urenenu listu objekata tipa XmlNode
POLJA (SVOJSTVA):
• **Count** – broj članova liste.
METODE:
• **Item**(int *index*) – vraća objekat tipa XmlNode koji nalazi na rednom mestu index u listi.

XmlAttribute objekat predstavlja atribut.
– Jedna od karakteristika DOM-a je način na koji se tretiraju atributi.
– Atributi nisu čvorovi, deca elementa, već se smatraju svojstvom elementa i sastoje se od imena i vrednosti.
XmlAttributeCollection klasa sadrži sve atribute odredjenog elementa.
– Svaki član ove kolekcije predstavlja **XmlAttribute** objekat.
POLJA (SVOJSTVA):
Count – broj elemenata u listi.

Append(XmlAttribute *node*)
– ubacuje atribut u kolekciju kao poslednji poslednji.
Prepend(XmlAttribute *node*)
– ubacuje atribut u u kolekciju kao prvi.
InsertAfter(XmlAttribute *newNode*, XmlAttribute *refNode*)
InsertBefore(XmlAttribute *newNode*, XmlAttribute *refNode*)
Item(int *index*)
– vraća XML atribut koji se nalazi na rednom mestu index u kolekciji.
Remove(XmlAttribute *node*)
RemoveAt(int *i*)
RemoveAll()
GetNamedItem(string *name*)

SOM

XML Schema (XSD)

Šema je XML dokument koji definiše klasu XML dokumenata specificiranjem njihove strukture.
Šema definiše dozvoljeni sadržaj XML dokumenta odnosno opisuje gramatiku koje dokumenti moraju da se pridržavaju da bi se smatrali validnim u odnosu na datu šemu.

Šema definiše strukturu, sadržaj i semantiku XML dokumenta.

Schema Object Model (SOM)
.NET Framework omogućava i manipulisanje XML šemama pomoću SOM-a.
omogućava čitanje, izmenu i validaciju XML šema programskim putem.
SOM radi sa XML šemama na sličan način kao što DOM radi sa konkretnim XML dokumentima.

System.Xml.Schema namespace
SOM se sastoji od velikog skupa klasa koje se nalaze u **System.Xml.Schema** namespace-u.
Ove klase korespondiraju osnovnim elementima XML šeme koji su propisani od strane W3C-a.
(schema, element, attribute, complexType, simpleType, attributeGroup, sequence, choice...)
Klase omogućavaju čitanje šeme iz datoteke ili drugog izvora kao i kreiranje šeme u memoriji, putem programa.
Po šemi se zatim može kretati, ona se može modifikovati, kompajlirati, validovati, ili upisati u datoteku.

XmlSchema klasa
Predstavlja memorijsku reprezentaciju XML šeme.
METODE:
Read(Stream *izvor,* ValidationEventHandler e)
Read(TextReader *izvor,* ValidationEventHandler e)
Read(XmlReader *izvor,* ValidationEventHandler e)
vraća objekat tipa XmlSchema koji predstavlja šemu učitanu iz zadatog izvora podataka.
Write(Stream *izvor)*
Write(TextWriter *izvor)*
Write(XmlWriter *izvor)*
piše datu šemu u zadati izvor podataka.

XmlSchema - svojstva
Elements - vraća kolekciju svih elementa date šeme.
objekti u kolekciji su tipa: XmlSchemaElement
Attributes - vraća kolekciju svih atributa date šeme.
objekti u kolekciji su tipa: XmlSchemaAttribute
Groups - vraća kolekciju svih grupa date šeme.
objekti u kolekciji su tipa: XmlSchemaGroup
AttributeGroups - vraća kolekciju svih globalnih grupa atributa date šeme.
objekti u kolekciji su tipa: XmlSchemaAttributeGroup
SchemaTypes - vraća kolekciju svih tipova date šeme.
objekti u kolekciji su tipa: XmlSchemaType, XmlSchemaSimpleType, XmlSchemaComplexType

Items - vraća kolekciju svih elemenata šeme.
Kolekcija objekata tipa:
XmlSchemaElement,
XmlSchemaGroup,
XmlSchemaAttribute,
XmlSchemaAttributeGroup,
XmlSchemaComplexType,
XmlSchemaSimpleType,
XmlSchemaAnnotation i
XmlSchemaNotation.

Kreiranje XmlSchema-e
Xml šeme se mogu kreirati programskim putem u memoriji korišćenjem sledećih klasa:
- **XmlSchemaElement**
- **XmlSchemaAttribute**

SVOJSTVA:
Name - definiše naziv datog elementa ili atributa.
SchemaTypeName

- definiše tip datog elementa ili atributa čije se odrenuje njihov sadržaj.
- tipovi mogu ugraneni ili korisnički definisani.
- za kreiranje novih korisnički definisanih tipova koriste se klase:

XmlSchemaSimpleType i **XmlSchemaComplexType**.

- Kreirani elementi se dodaju u šemu preko svojstva šeme **Items**.
- Na kraju je neophodno da se data šema kompajlira korišcenjem **Compile** metode klase **XmlSchemaSet.**
- vrši se validacije programski izgradene šeme.
- vrši se verifikacija semanticke ispravnosti šeme.

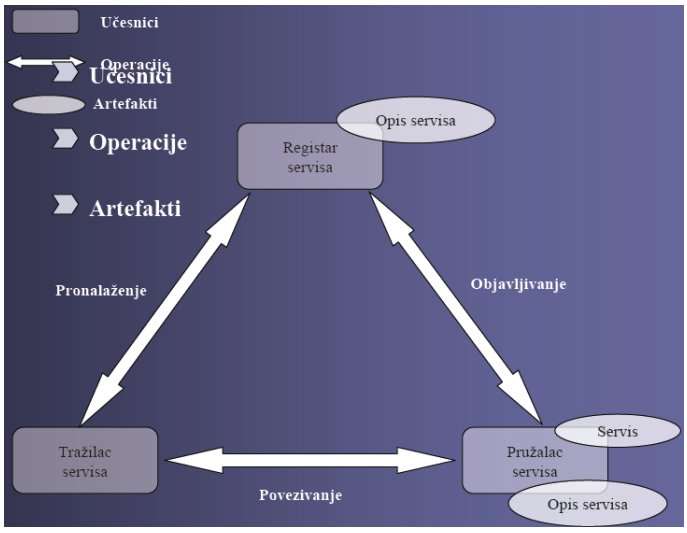
```
XmlSchemaSet schemaSet = new XmlSchemaSet();
schemaSet.Add(kreiranaSema);
schemaSet.Compile();
```

14. Web servisi

Web servisi – Cilj
Omogućiti povezivanje poslovanja (Business to Business) odnosno, omogućiti programsko povezivanje distribuiranih softverskih komponentibez obzira na kojoj su platformi realizovani, koji je programskijezik tom prilikom korišćen, kao i platforma na kojoj se izvršavaju. Dva zahteva:
Reusability & Interoperability
Web servisi – Vizija
Postojanje miliona nezavisnih komponenata dostupnih preko Interneta koje su upotrebljive na bilo kojoj platformi i svim razvojnim jezicima.

Definicije Web servisa
Web servisi su modularne, samoopisujuće aplikacije koje se mogu objaviti,locirati i pozvati sa bilo koje tačke Web-a ili lokalne mreže.
Web servisi su nova platforma za izgradnju interoperabilnih distribuiranih aplikacija. Ona predstavlja skup standarda koje aplikacije moraju da poštuju kako bi se postigla interoperabilnost preko Web-a.
Web servisi su distribuirane softverske komponente koje su dostupne kroz standardne Internet protokole.

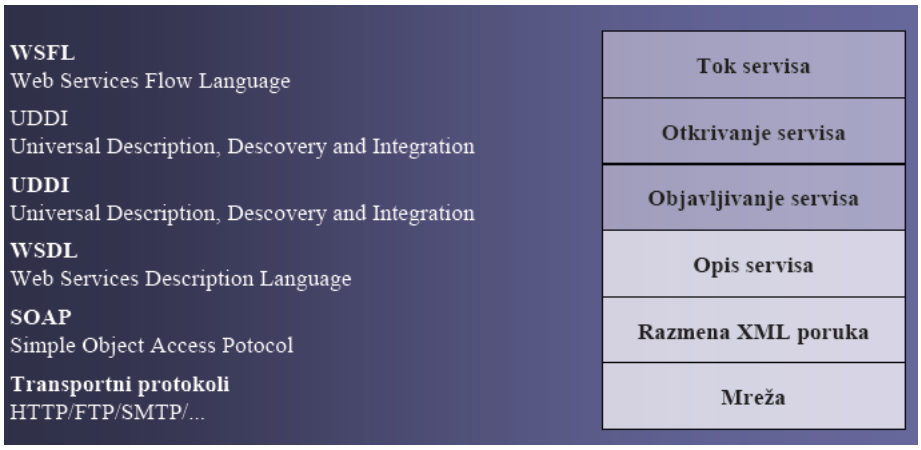
Model Web servisa



Arhitektura Web servisa

Interoperabilnost

Da bi se postigla interoperabilnost, neophodan je stek Web servisa kojim se definišu standari na svakom nivou



Osnova bez koje se ne može

Poslednja tri sloja steka Web servisa su neophodna da bi se obezbedio ili koristio bilo koji Web servis.

Stek interoperabilnosti Web servisa

Simple Object Access Protocol nb SOAP

SOAP je komunikacioni protokol, baziran na XML-u, za razmenu informacija između računara bez obzira na njihov operativni sistem, programsko okruženje ili objektni model.

U okviru specifikacije, definisan je kao jednostavan protokol za razmenu informacija između uređaja ravnopravnih komunikacionih mogućnosti u decentralizovanom i distribuiranom okruženju kao što je Internet.

SOAP nije transportni protokol.

SOAP poruka

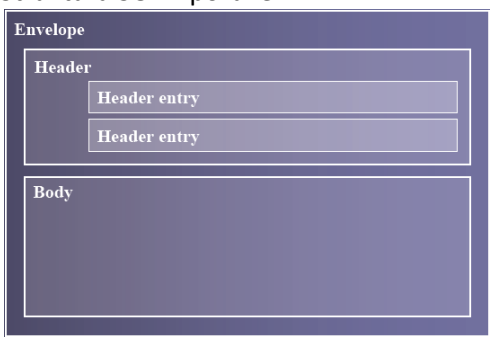
SOAP poruka je XML dokument

Osnovni delovi SOAP poruke su-element Envelope-element Header-element Body

SOAP poruke predstavlju sredstvo preko kojih aplikacije komuniciraju

Poznavanjem ovih elemenata znamo gde se smeštaju podaci, kako se poruka proširuje i kako se predstavlja greška.

Struktura SOAP poruke

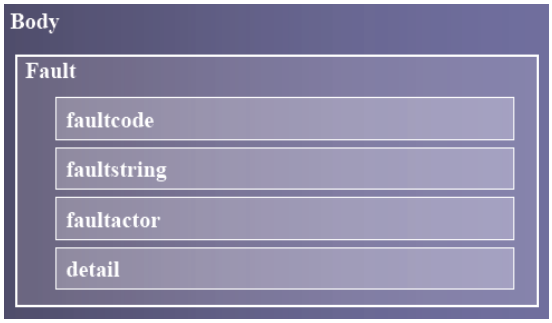


```
<Envelope ...>
        <Header>
                ...
        </Header>
        <Body>
                ...
        </Body>
</Envelope>
```

Prijavljivanje greške greške

Prilikom razmene SOAP poruka može nastati greška

```
<Envelope ...>
        <Body>
                <Fault >
                        <faultcode> ... </faultcode>
                        <faultstring> ... </faultstring>
                        <faultactor> ... </faultactor>
                        <detail> ... </detail>
                <Fault >
        </Body>
</Envelope>
```



SOAP kodiranje

Predstavljanje podataka unutar SOAP poruke naziva se SOAP kodiranje

Odnosi se na podatke koji se nalaze unutar Body elementa

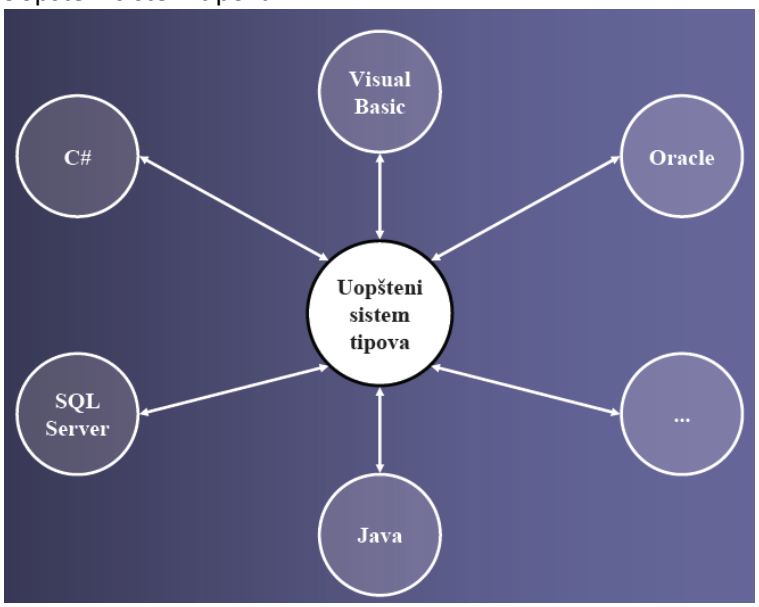
Zašto je važno kodiranje ?

S obzirom da su poruke sredstvo preko kojih aplikacije komuniciraju, neophodno je da one budu formirane na način koji im je razumljiv. Ovo se pre svega odnosi na aplikacije koje primaju poruku.

-da li je 1234 broj ili običan tekst ?

-što je tačno integer (16, 32 ili 64 bita) ?

Uopšteni sistem tipova



Web Services Description Language WSDL

Kako potencijalnom korisniku objasniti koje funkcije Web servis izlaže i koje parametre svaka ta funkcija prihvata?
Da li je SOAP dovoljan za to?
Primer SOAP poruke kojim se ilustruje poziv Web servisa.Poređenje: poziv funkcije C++ biblioteke na osnovu nekog primera, bez posedovanja header datoteke.
Rešenje je WSDL. Njime se definišu i opisuju Web servisi, baškao što header datoteke definišu i opisuju tradicionalne binarne datoteke. WSDL je gramatika bazirana na XML-u za opisivanje Web servisa.
Opis servisa pokriva sve detalje koji su neophodni za komunikaciju sa servisom, što uključuje format poruke, transportne protokole i lokaciju servisa.

Element Types

Koristi se za opisivanje tipova ili struktura podataka koji će se koristiti u porukama
Strukture podataka opisane ovim elementom su apstraktni tipovi
Sintaksa za definisanje elementa types

```
...
<types>
        ...
</types>
...
```

Element Message

Sastoji se od jednog ili više elemenata part, pri čemu svaki taj element part ukazuje na određeni tip koji je definisan elementom types.
Nazivi message elemenata su proizvoljni, ali se unutar WSDL dokumenta ne smeju pojaviti dva sa istim nazivom. Ovo isto važi i za part elemente.
Elementi part se koriste za logičko razdvajanje podataka unutar poruke

Sintaksa za definisanje elementa message

```
...
  <message name=“nmtoken”>
        <part name=“nmtoken”element=“qname”| type=“qname”/>
  </message>
...
```

WSDL elementi

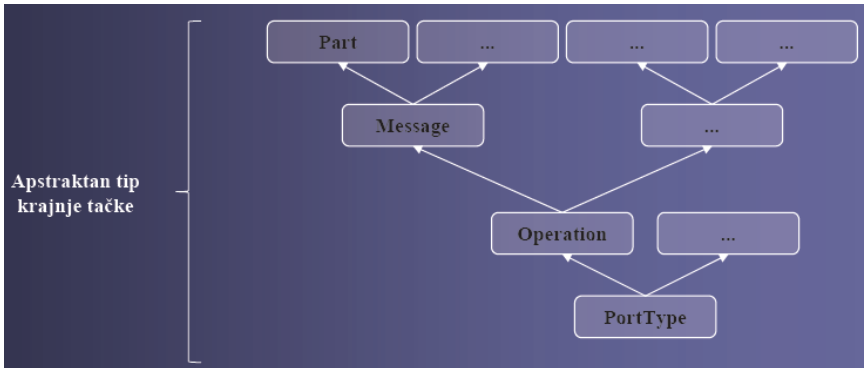
Element PortType

Predstavlja imenovani skup apstraktnih operacija i obuhvaćenih apstraktnih poruka, odnosno skup poruka koje su grupisane u operacije, pri čemu operacija predstavlja jedinicu rada servisa.
Svaku operaciju mogu da čine input, output i fault poruka, a to koje će se od njih pojaviti zavisi od tipa operacije.
Ove operacije se mogu porediti sa metodama. (ulazni i izlazni parametri metode –input i output elementi)

Sintaksa za definisanje elementa PortType

```
  <portType name="nmtoken">
   <operation name="nmtoken">
     <input name="nmtoken" message="qname"/>
     <output name="nmtoken" message="qname"/>
        <fault name="nmtoken" message="qname"/>
   </operation>
  </portType>
```

WSDL elementi



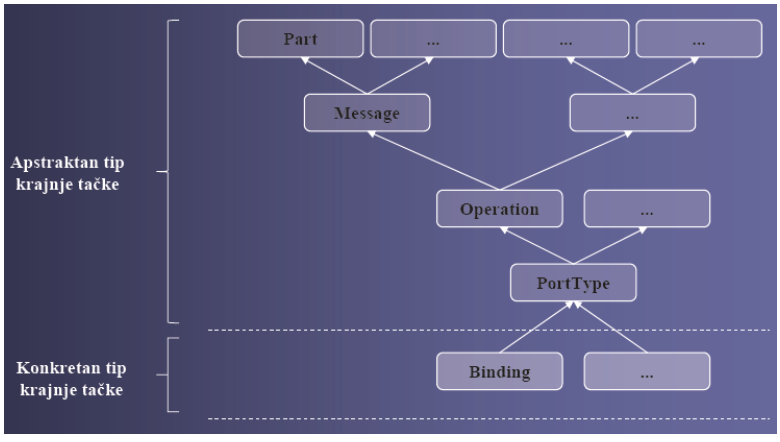
Element Binding

Elementom Binding se definiše kako je operacija povezana sa određenim protokolom, odnosno definiše se format poruke i detalji vezani za protokol.

Sintaksa za definisanje elementa Binding je slična sintaksi elementa PortType, ali se svaka operacija prevodi u operaciju za određeni protokol.

```
...
<binding name="nmtoken" type="qname">
        ...
        <operation name="nmtoken">
            ...
            <input name="nmtoken">
                ...
            </input>
            <output name="nmtoken">
                ...
            </output>
            <fault name="nmtoken">
                ...
            </fault>
        </operation>
</binding>
...
```

WSDL elementi



Element Port

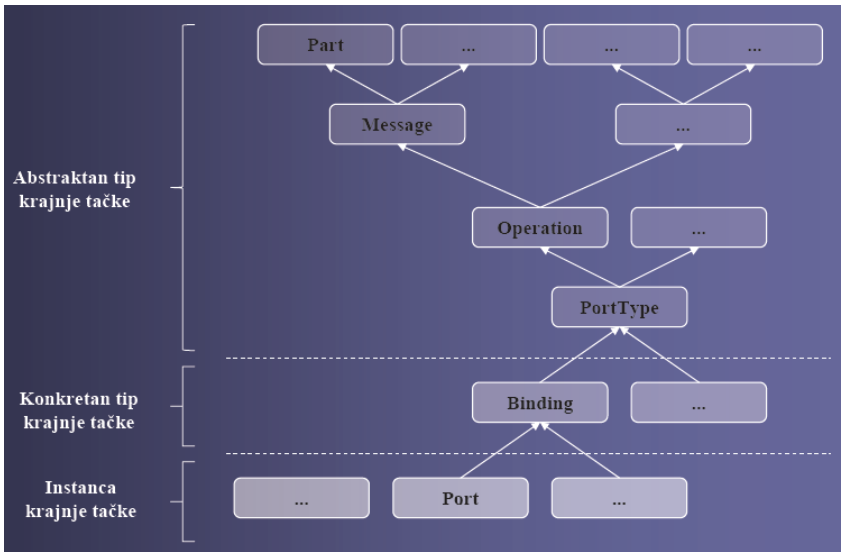
Element Port povezuje definisani element Binding sa adresom, čime se zapravo predstavlja stvarna mrežna krajnja tačka odnosno stvarne mrežne krajnje tačke na kojima servis komunicira.

Svaki Port ukazuje na prethodno definisani element Binding i na samo jednu adresu.

Sintaksa za definisanje elementa Port

```
...
<service ...>
        <port name=''nmtoken'' binding=''qname''>
                ...
        </port>
</service>
...
```

WSDL elementi



Element Service

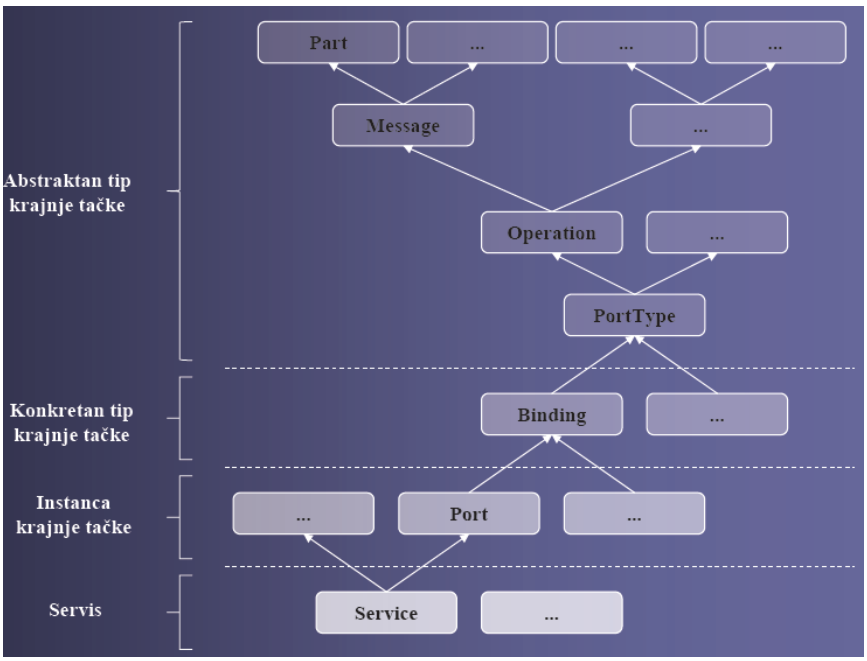
Element servis predstavlja kolekciju elemenata Port.

Servis može da sadrži više port-ova koji koriste isti portType, ali imaju različita povezivanja i/ili adrese. U tom slučaju portovi predstavljaju alternative. Korisnik vrši izbor na osnovu protokola za komunikaciju ili na osnovu udaljenosti adrese.

Sintaksa za definisanje elementa PortType

```
...
<service name="nmtoken">
        <port ...>
        </port>
</service>
...
```

WSDL elementi



Primer Web servis Kursna lista

Projektni zahtev

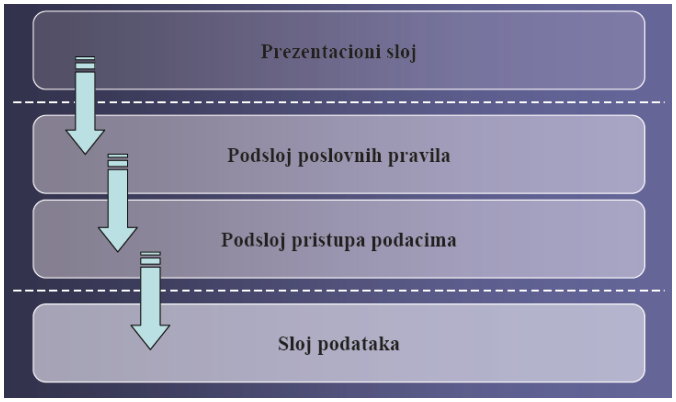
Potrebno je kreirati Web servis Narodne banke Jugoslavije kojim se omogućava da potencijalni korisnici, preko Interneta dobiju sve potrebne informacije u vezi sa zvaničnom kursnom listom. Informacije se odnose na srednji, prodajni i kupovni kurs određene valute.

Pored ovih informacija, Web servis treba da omogući i uvid u iznos koji se dobija prilikom konverzije iz jedne u drugu valutu, kao i uvid u proviziju koja se tom prilikom naplaćuje.

Znači, ovim Web servisom potrebno je omogućiti da se deo poslovanja Narodne banke Jugoslavije izloži na Internet.

Posle kreiranja, Web servis oglasiti na adresi ''http://www.nbj.com/servisi/kursnalista''. Ovo će biti lokacija na kojoj će potencijalni korisnici moći da komuniciraju sa Web serisom.

Korišćena arhitektura



Infrastruktura Web servisa – Kursna lista