

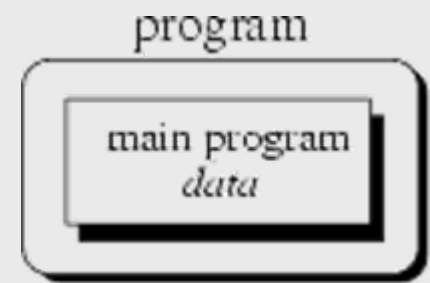
Tehnike programiranja i standardi kodiranja

4/13

Tehnike programiranja

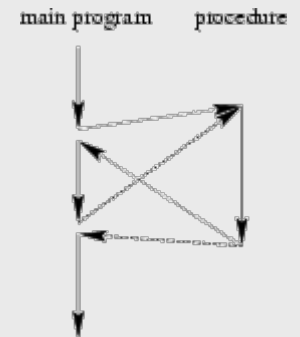
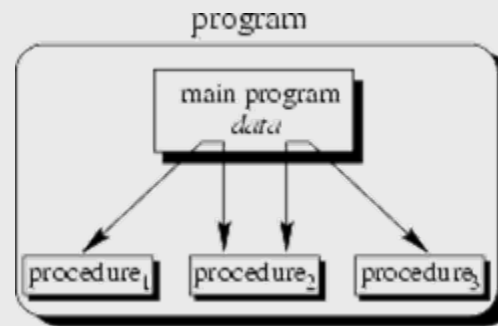
❑ Nestrukturirano

- Programi koji se sastoje od slijeda naredbi i djeluju nad zajedničkim skupom podataka (globalnim varijablama)
- Ponavljanje nekog posla znači i kopiranje naredbi.



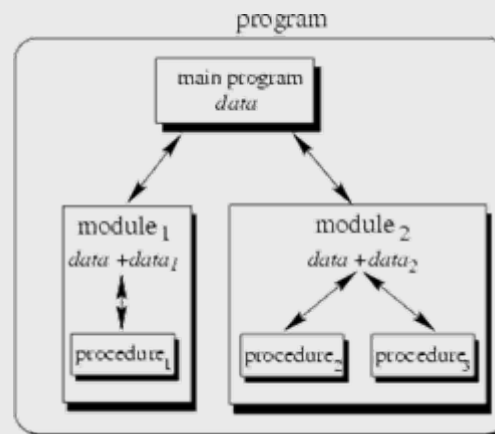
❑ Proceduralno

- Izdvajanjem naredbi u procedure,
- Program postaje slijed poziva procedura.



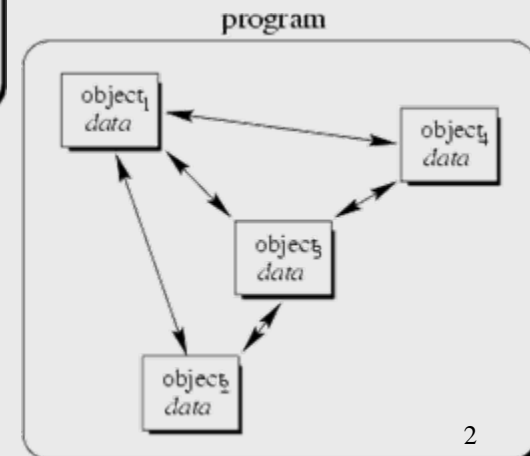
❑ Modularno

- Procedure zajedničke funkcionalnosti grupiraju se u module
- Dijelovi komuniciraju pozivima, moduli mogu imati lokalne podatke



❑ Objektno

- Objekti komuniciraju slanjem poruka



Dijelovi programa

❑ programska cjelina (program unit)

- skup programskih naredbi koje obavljaju jedan zadatak ili jedan dio zadatka, npr. glavni program, potprogrami (procedure, funkcije)

❑ programski modul

- skup logički povezanih programskih cjelina → modularno programiranje
- npr. C datoteka sa statičkim varijablama

❑ komponenta

- bilo koji sastavni dio softvera, uobičajeno podrazumijeva fizičke cjeline
- npr. assembly (skup, sklop)
 - DLL ili EXE (izvršna) datoteka uključujući pripadajuće resursne (datoteka.resx) ili sigurnosne elemente

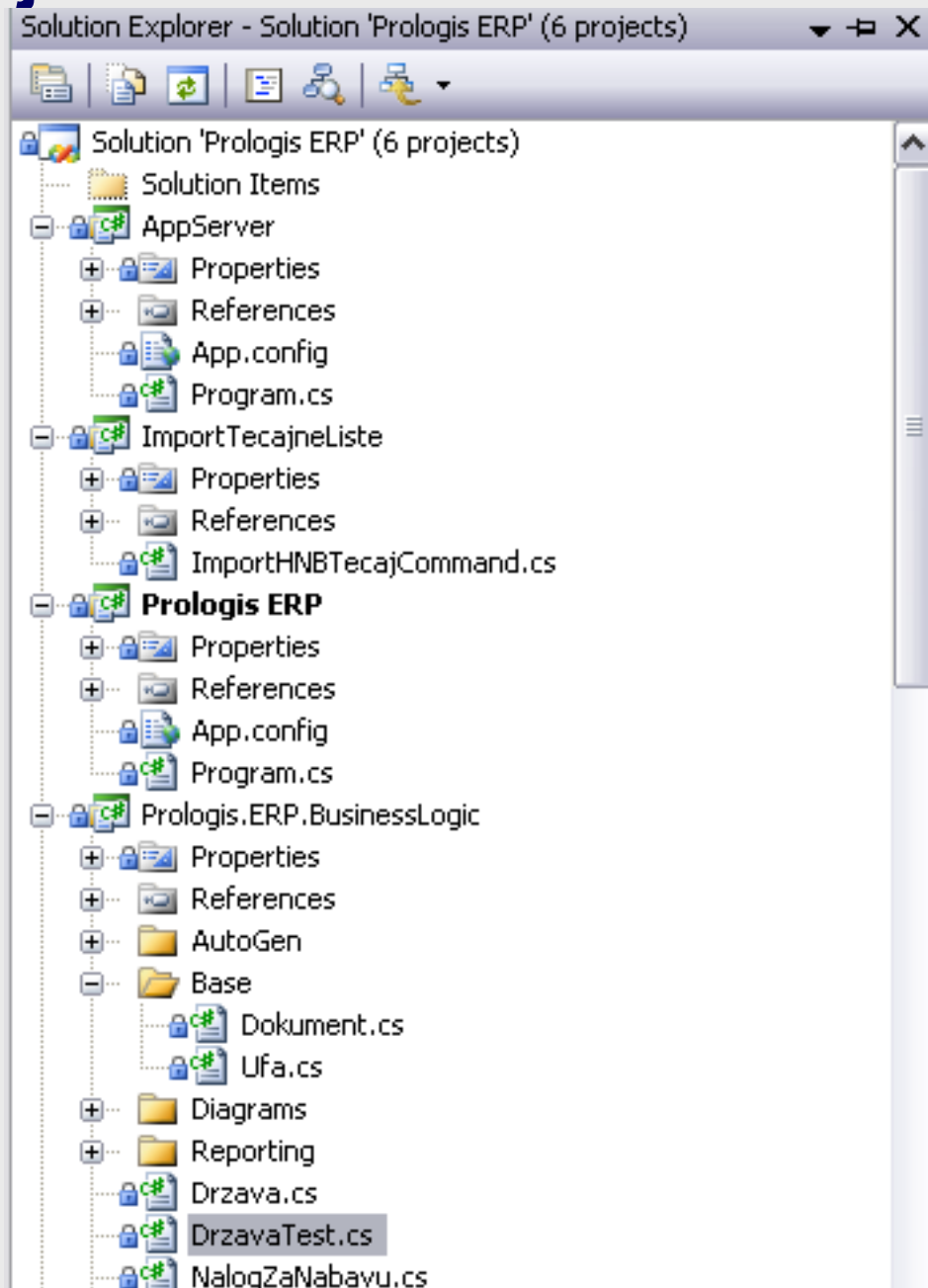
Organizacija datoteka

❑ Izbjegavati preduge datoteke

- datoteke s više od 300-400 programskih redaka restrukturirati (refactor) uvođenjem pomoćnih (helper) razreda
- najbolje je staviti po jedan razred u zasebnu datoteku, naziva jednakog nazivu razreda

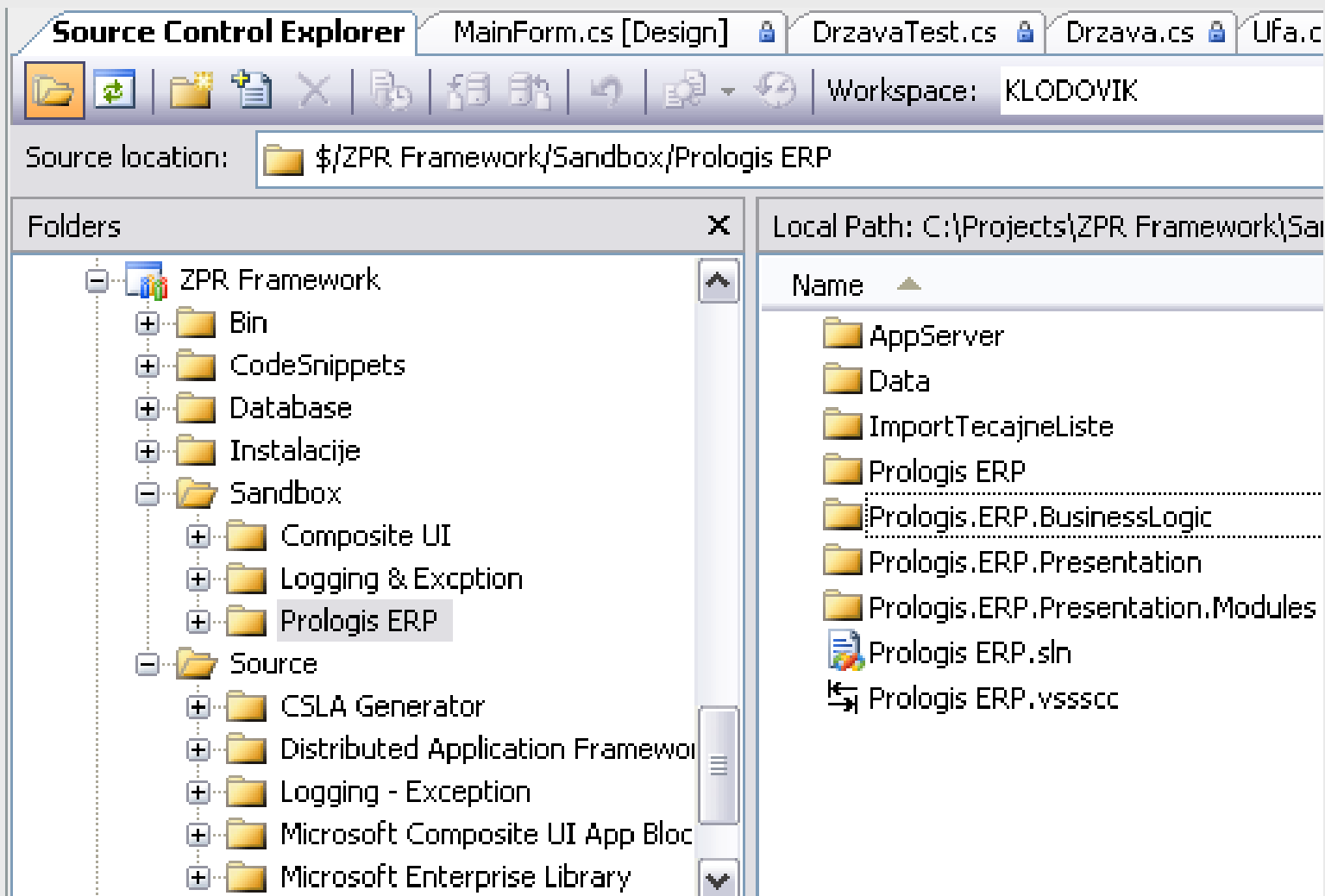
❑ Komponente imenovati sukladno hijerarhiji

- <Company>.<Component>.dll, npr. Prologis.ERP.BusinessLogic.dll
- Menus.FileMenu.Close.Text



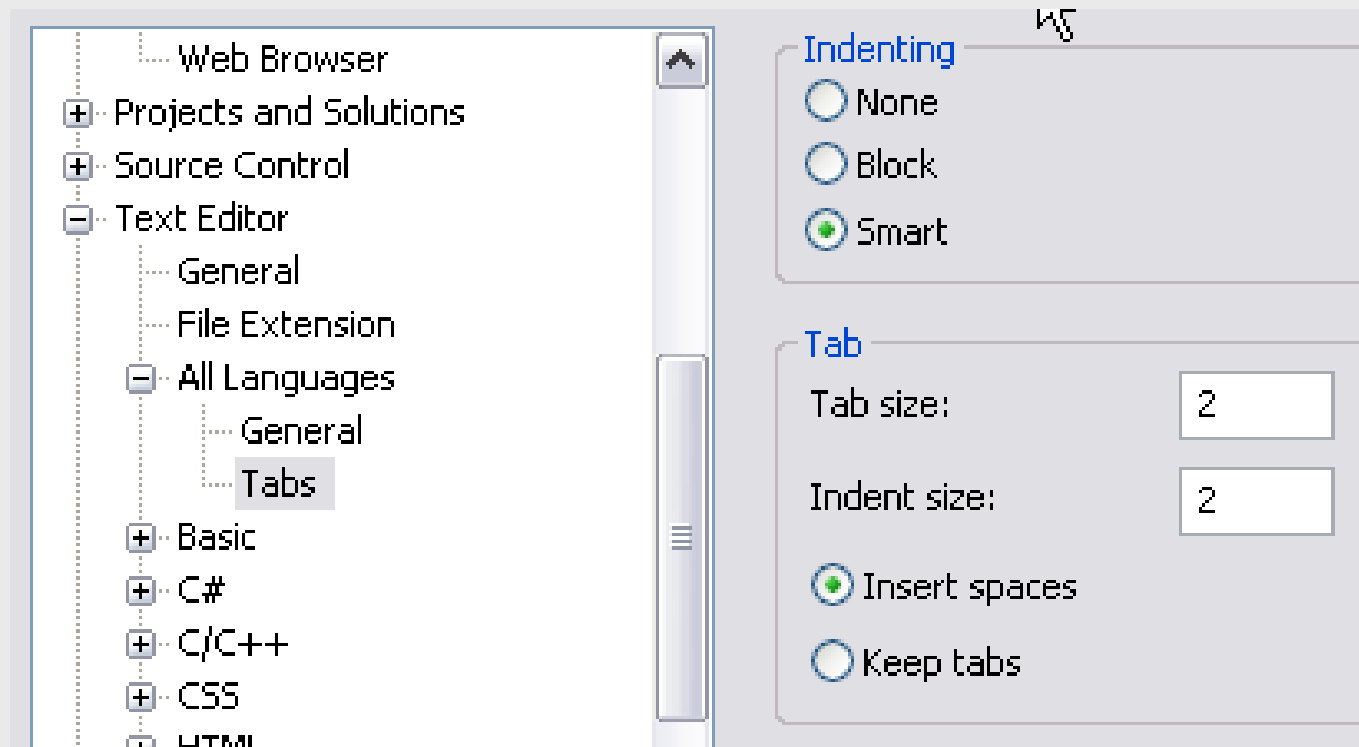
Organizacija datoteka

- ❑ Organizacija datoteka treba slijediti organizaciju komponenti



Formatiranje izvornog programskog koda

- različito označavanje elemenata jezika, kao što su rezervirane riječi, identifikatori, komentari, opcije prevoditelja (različita boja)
- izbjegavanje redaka koji duljinom prelaze širinu zaslona (80 znakova)
- pisanje najviše po jedne programske naredbe u retku
- podjela slijedova naredbi na odsječke koji su u cjelini vidljivi na zaslonu
- tzv. "uvlačenje" (*indentation*) kôda unutar programskih struktura



Formatiranje bloka naredbi

- ❑ Redak proreda za odvajanje logičkih grupa naredbi
- ❑ Vitice u zaseban redak poravnate s naredbom strukture
- ❑ Preporuka vitica i onda kada blok nema ili ima samo jednu naredbu

```
bool VratiPare(string name)
{
    string fullMessage = "Vrati pare " + name;
    DateTime currTime = DateTime.Now;
    fullMessage += fullMessage + " u " + currTime.ToString();
    MessageBox.Show(fullMessage);

    if (showResult == true)
    {
        for (int i = 0; i < 10; i++)
        {
            // radi
        }
    }

    return true;
}
```

Formatiranje naredbe ili izraza

- ❑ **Jedan razmak između operatora, ali ne uz zagrade**
- ❑ **Postupanje kada naredba ili izraz premašuju poželjnu širinu retka**
 - Prekid (skok u novi redak) na separatoru liste, tj. zarezu.
 - Prekid prije operatora
 - Bolje prelomiti izraz na višoj razini nego onaj na nižoj razini
 - Poravnanje narednog retka s početkom izraza u prethodnom retku
- ❑ **Primjeri**

```
longMethodCall(expr1, expr2,  
                expr3, expr4, expr5);
```

```
var = a * b / (c - g + f)  
      + 4 * z;
```


Standardizacija nazivlja

□ Notacije za obilježavanje programskih objekata

- *PascalCase* - početno slovo svake riječi u imenu je veliko slovo
 - npr. `BackColor`
 - koristi se kod imenovanja prostora imena, razreda, sučelja, pobrojanih tipova, postupaka i svojstava, te public atributa
 - identifikator razreda može započeti znakom `@`
- *camelCase* – početno slovo prve riječi u imenu je malo slovo, početna slova ostalih riječi u imenu su velika slova
 - npr. `backColor, jedinstveniMaticniBroj`
 - koristi se kod zaštićenih atributa i lokalnih varijabli postupaka
- Preporuke
 - Imena sučelja uobičajeno započinju slovom `I`
 - Koristiti imenice za imena razreda
 - Koristiti glagolske oblike za imena postupaka

Primjer naziva programskih elemenata

- ❑ **PascalCase** za imena razreda i postupaka
- ❑ **camelCase** za lokalne varijable i argumente potprograma

```
public class Ispit
{
    int totalCount = 0;

    void Prijavi(string jmbag)
    {
        ...
        string fullMessage = "Prijavljen " + jmbag;
        ...
    }
}
```

Smjernice za nazivlje

□ Nazivlje struktura podataka

- pridjeljivati nazive iz kojih se vidi na što se odnose
 - Primjerice: Osoba.SifraOsobe, Mjesto.SifraMjesta
 - Umjesto: Osoba.Sifra, Mjesto.Sifra, Artikl.Sifra
- izbjegavati uporabu posebnih znakova koje sintaksa jezika/sustava ne dozvoljava pri tvorbi identifikatora
 - pr. operatori i znakovi za palatale našeg jezika, Dat-Rođ
- izbjegavati prekratke nazive koji, osim u nečitkost, vode u nedosljednost već pri prvoj pojavi iste kratice za različiti pojam
 - npr. SifMje za Mjeru i Mjesto
- izbjegavati preduge nazive, pr.
UvoznaCarinskaDeklaracija.RedniBrojStavkeKalkulacije, zbog
 - smanjenja čitljivosti
 - učinkovitosti ručnog kodiranja (sintaksne pogreške izazvane tipkarskim)
 - mogućih ograničenja jezika (pr. duljina identifikatora do 18, 30... znakova)
- izbjegavati nazive dobivene rutinskim spajanjem naziva entiteta i atributa jer mogu djelovati nezgrapno unutar upita, na primjer:
 - umjesto `SELECT Posao.* FROM Posao WHERE Posao.posao_datum ...`
 - bolje je `SELECT Posao.* FROM Posao WHERE Posao.DatPosla ...`
- koristiti nazive koji se daju izgovoriti
 - pr. Nstvnk.SifNstvnk → Nastav.SifNastav ili Nastavnik.SifNastavnika

Smjernice za nazivlje

□ Nazivlje programskih varijabli

- koristiti smislene nazive
 - izbjegavati "jednoslovčane" varijable, pr. i, j, k ili i, ii, iii, ili, x1, x2, x3
 - osim za indekse i dimenzije polja, pr. i, n
- nazive odabirati u skladu sa značenjem sadržaja
 - pr. max za najveću vrijednost
 - pr. len za varijablu koja određuje duljinu
- koristiti standardne prefikse/sufikse za srodne elemente/objekte
 - pr. frmOsoba ili fOsoba za zaslonsku masku
 - pr. repOsoba, rOsoba za izvješće ...
- dozvoljeno koristiti jednoznačne kratice općih pojmova kao što su
 - pr. broj, redni broj, šifra, kratica, oznaka, datum
 - → br, rbr, sif, krat, ozn, dat

Deklaracija varijabli

❑ Lokalne varijable

- izbjegavati deklaraciju više varijabli u istom retku zbog komentiranja
- inicijalizaciju provesti na mjestu deklaracije

```
int level; // razina rekurzije  
int size; // dimenzije matrice  
  
int a, b; //monolog ili recitacija ?
```

```
string name = mojObjekt.Ime;  
int hours = vrijeme.Sati;
```

❑ Članske varijable

- deklarirane na vrhu razreda
- privatne
- nad njima javna/zaštićena svojstva

```
class Osoba  
{  
    private string jmbg;  
    public string JMBG {  
        get { return jmbg; }  
        set { jmbg = value; }  
    }  
}
```

Inicijalizacija varijabli

- ❑ Inicijalizaciju provesti na što jednostavniji način

```
// v1
bool pos;
if (val > 0)
{
    pos = true;
}
else
{
    pos = false;
}
```

```
// v2
bool pos = (val > 0) ? true : false;
```

```
// v3
bool pos;
pos = (val > 0);
```

```
// v4
bool pos = (val > 0);
```

Inicijalizacija varijabli

❑ Inicijalizacija polja referenci

```
public class Krumpir
{ }

public class PoljeKrumpira
{
    const int VelicinaPolja = 100;
    Krumpir[] polje = new Krumpir[VelicinaPolja];

    PoljeKrumpira()
    {
        for (int i = 0; i < polje.Length; ++i)
        {
            polje[i] = new Krumpir();
        }
    }
}
```

Metode

❑ Metoda

- naziv metode mora upućivati na to što metoda radi
- metode s više od 25 redaka preoblikovati podjelom u više metoda
- izbjegavati metode s više od 5 argumenata – koristiti strukture

```
void SavePhoneNumber (string phoneNumber)
{
    // spremi fon
    ...
}
```

```
// prema broj telefona
void SaveData (string phoneNumber)
{
    // spremi fon
    ...
}
```


Metoda mora obavljati samo jedan posao

```
// spremi adresu
SaveAddress (address);

// posalji mail obavijesti promjene
SendEmail (address, email);

void SaveAddress (string address)
{
    // spremi
    // ...
}

void SendEmail(string address,
               string email)
{
    // posalji
    // ...
}
```

```
// spremi i salji
// mail obavijesti
SaveAddress(address, email);
void SaveAddress(
    string address,
    string email)
{
    // 1: spremi
    // ...
    // 2: salji
    // ...
}
```

Komentari

❑ Preporuke

- paziti da komentari budu ažurni, tj. da odgovaraju stvarnom stanju
- ne pretjerivati u pisanju komentara
- loš kôd bolje je iznova napisati, nego (bezuspješno) pojašnjavati
- komentirati smisao naredbi (izbjegavati "prepričavanje")

❑ Komentari bloka trebaju biti na istoj razini gdje i kôd

❑ Izbjegavati blok komentare i “staromodne” retkovne C komentare

```
/******\
* windowsx.h - Macro APIs, window message crackers, *
*                                     and control APIs *
*                                     Version 3.10 *
\*****/

/* slijede neke f-je */
```

Primjer komentara

- ❑ Opis, argumenti, trag izmjena, retkovni / komentar bloka
- ❑ Oznake XML oblika za generiranje dokumentacije

```
/// <summary>
/// Execute a SqlCommand (that returns ...)
/// </summary>
/// <remarks>
/// e.g.:
///     int result = ExecuteNonQuery(connectionString, ...
/// </remarks>
/// <param name="connectionString">A valid string</param>
...
/// <returns>An int representing ...</returns>
/// <created></created>
/// <modified>K.Fertalj, 25.11.2007. 00:13</modified>
public static int ExecuteNonQuery(string connectionString,
                                CommandType commandType, string commandText)
{
    // Pass through the call ...
    return ExecuteNonQuery(connectionString,
                           commandType, commandText, (SqlParameter[])null);
}
```

Konstante, skraćenice, tipovi podataka

- ❑ `const` koristiti samo za stvarno nepromjenjive vrijednosti
- ❑ Koristiti sva velika slova u nazivu varijable ukoliko predstavlja kraticu naziva dvije do tri riječi ili jest duljine do tri znaka
- ❑ Koristiti tipove podataka definirane programskim jezikom, a ne sistemske (definirane u imeniku `System`)
 - `object`, `string`, `int` (ne `Object`, `String`, `Int32`)

```
public class MojaMatka
{
    private const short BrojDanaTjedna = 7;
    public const double PI = 3.14159;
    public const float PDV = 0.22f;
}
```

```
enum MailType
{
    Html,
    PlainText,
    Attachment
}

void SendMail(string message,
               MailType mailType)
{
    switch(mailType)
    {
        case MailType.Html:
            // radi html
            break;
        case MailType.PlainText:
            // radi txt
            break;
        case MailType.Attachment:
            // radi attach
            break;
        default: // uvijek default!
            // radi ostalo
            break;
    }
}
```

❑ Izbjegavati izravnu upotrebu konstanti (hard kodiranje)

- koristiti pobrojane tipove, predefinirane konstante i datoteke s parametrima i resursima

```
void SendMail(string message,
               string mailType)
{
    switch (mailType)
    {
        case "Html":
            // radi html
            break;
        case "PlainText":
            // radi txt
            break;
        case "Attachment":
            // radi attach
            break;
        default:
            // radi ostalo
            break;
    }
}
```

Pobrojane vrijednosti

❑ Izbjegavati eksplicitni navod vrijednosti

```
// Dobar
public enum Barjak
{
    Crven, Bijeli, Plavi
}

// Manje dobar
public enum Barjak
{
    Crven = 1, Bijeli = 2, Plavi = 3
}
```

Logički izrazi

❑ Poziv funkcija u logičkim izrazima

```
book ok = IsItSafe();  
if (ok) // umjesto if (IsItSafe())  
{  
    ...  
}
```

❑ Usporedba s true/false

```
while (condition == false) // loš  
while (condition != true) // loš  
while (condition) // OK
```

❑ Usporedba realnih brojeva

- izbjegavati operatore == i !=
- osim kada je varijabla inicijalizirana na 0 (1, 2, 4, ...) pa se kasnije želi utvrditi da je došlo do promjene

Nazivi objekata u bazi podataka

tablica	jednina, bez prefiksa/sufiksa	Osoba, Racun
tablica	spojne kao RoditeljDijete	ArtiklRabat, PartnerKontakt
polje	ŠtojeEntitetu	SifraKupca, BrojStavki
polje	IdTablice za jednoznačni identifikator	IdEvidencije
polje	bool s prefiksom Ima, Je, Zast	ImaPovijest, JeZapisan, ZastZakljucan
polje	smisao broja, zbirna imenica	RadniSati, RadnoVrijeme
ključ	pk_Tablica	pk_Osoba, pk_RadnoMjesto
ključ	fk_Roditelj_Dijete, fk_Roditelj_Uloga	fk_Osoba_MjestoRod, fk_Osoba_IdMjestaRod
ključ	ix_Tablica_Polje	ix_AutorVrste_IdVrste
ključ	ux_Tablica_Polje	ux_Tecaj_OznValutaDatTec
check	ck_Tablica_Polje_KojiCheck	ck_Stavka_Kolicina_Poz, ck_Stavka_Kolicina_Max
default	df_Tablica_Polje	df_Racun_Rabat
pogled	ww_Pogled	ww_MojiProjekti
proc	ap_StoraOpcenita	ap_Help
proc	fn_FunkcijaOpcenita	fn_PronadjiProizvode
proc	ap_Tablica_CRUDQX # kombinacija	ap_UlaznaFaktura_C
trigger	tr_Tablica_IUD	tr_Artikl_I

Stupci tablica u bazi podataka

Polje	Kratica	Tip	Duljina
Sifra	Sif	int	4
Broj	Br	int	4
Redni broj	Rbr	int	serial
Kolicina	Kol	decimal	18,4
Zastavica (flag)	Zast	bit	1
Status	Stat	varchar	2
Iznos	Izn	decimal	18,4

Primjer nekih kratica u stvarnom projektu

Automatski	Auto	
Cijena	Cij	
Datum	Dat	
Devize	Dev	
Generiranje	Gener	??
Grupa	Gr	
Knjizenje	Knjiz	
Koeficijent	Koef	
Način	Nac	
Namjena	Namj	??
Obračun	Obrac	
Osnovica	Osnov	??
Placanje	Plac	
Početak	Poc	
Porezni, Poreznog	Porez	
Pozicija	Poz	
Skladište	Sklad	??
Vrijednost	Vrij	
Vrsta	Vr	
Završetak	Zav	

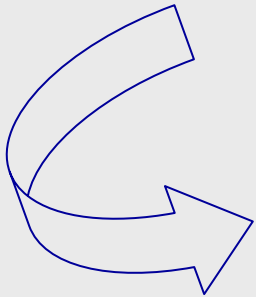
Tehnika i stil programiranja

❑ Tehnika i stil programiranja

- postaviti početne vrijednosti varijabli prije uporabe
- ugraditi podrazumijevane (default) vrijednosti ulaznih podataka (pdv = 22%)
- provjeravati zahtijevanost i valjanost ulaznih podataka
- dosljedno formatirati podatke (npr. datum, novac, broj)
- pripaziti na granične vrijednosti podataka, indeksiranih varijabli ...
- provjeriti moguće numeričke pogreške (množenje, dijeljenje s malim brojem)
- izbjegavati usporedbu na jednakost brojeva s pomičnim zarezom
- koristiti zagrade radi naglašavanja redoslijeda izračunavanja izraza
- presložiti i pojednostavniti nerazumljive izraze
- izbjegavati nepotrebna grananja
- izbjegavati trikove (ne žrtvovati jasnoću radi "efikasnosti")
- ponavljajuće blokove i izraze zamijeniti potprogramima
- rekurziju koristiti samo za rekurzivne strukture podataka
- prvo napraviti jasno i ispravno rješenje, a zatim brzo, lijepo ... rješenje
- neučinkoviti kôd ne usavršavati, nego naći bolji algoritam

Neki primjeri

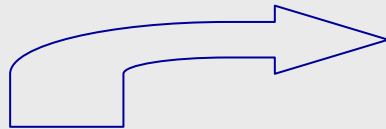
```
if ( ( a && !c ) || ( a && b && c ) ) {  
    rezultat = 1;  
} else if ( ( b && !a ) || ( a && c && !b ) ) {  
    rezultat = 2;  
} else if ( c && !a && !b ) {  
    rezultat = 3;  
} else {  
    rezultat = 0;  
}
```



```
static int rezultatTable[ 2 ][ 2 ][ 2 ] = {  
    // !b!c !bc b!c bc  
    0, 3, 2, 2, // !a  
    1, 2, 1, 1 // a  
};  
...  
rezultat = rezultatTable[ a ][ b ][ c ];
```

Primjer Kodiranje\StandardiKodiranja

❑ Primjer 1



```
if ( ( a && !c )  
    || ( a && b && c ) ) {  
    ...  
}
```

```
rez = ( a && !c ) ;  
rez |= ( a && b && c ) ;  
if (rez)  
{ ...  
}
```

❑ Primjer 2

```
float f = 3.14f;  
bool b = false;  
if (f == 3.14)  
{  
    b = true;  
}  
// vrijednost b ?  
// kada bi bila drukčija ?
```

❑ Primjer 3

```
f = 10 * 0.01f;  
if (f == 0.1)  
{  
    b = true;  
}  
// vrijednost b ?  
// kada bi bila drukčija ?
```

Organizacija programskog koda

❑ Programske knjižnice s funkcijama grupiranim po namjeni

- Održavanje, modularnost, opetovana iskoristivost , lokalizacija, ...
- funkcije za rad s općim tipovima podataka (npr. nizovi znakova i datumi)
- funkcije za rad s podacima u bazi podataka (npr. funkcije za upravljanje transakcijama i provjeru statusa izvedenih upita)
- funkcije sučelja (npr. sustav izbornika, poruka i pomoći)
- funkcije za održavanje baze podataka (npr. provjera konzistentnosti podataka i izrada rezervnih kopija)
- funkcije za administriranje vanjskih uređaja (npr. terminali i pisači)
- programski dio sustava zaštite (npr. definiranje programskih modula, funkcija i korisnika te rukovanje pravima pristupa programima i podacima)

❑ Napraviti provjere i inicijalizaciju pri pokretanju programa

- Konfiguracijske datoteke
- U nedostatku konfiguracije pokrenuti s predviđenim (default) vrijednostima

❑ Suvisle poruke i sugestija popravka

- “Pristup bazi podataka nije moguć. Moguće neispravno ime ili zaporka.”
umjesto “Pogreška”
 - u pozadini zapisati trag izvršenja radi dijagnostike problema

Defenzivno programiranje

Tehnike obrade pogrešaka

Defenzivno programiranje

❑ **Zaštita programa od neispravnog unosa podataka**

- Defenzivna vožnja automobila temelji se na načelu da vozač nikad ne može biti siguran što će učiniti drugi vozači, pa unaprijed nastoji izbjeći nezgodu za slučaj pogreške drugih vozača
- u defenzivnom programiranju ideja vodilja je da će potprogram s neispravnim podacima “opstati” i onda kada su pogreškom pozivajuće procedure predani neispravni argumenti
- Pristup “smeće unutra, smeće van” (“garbage in, garbage out”) treba zamijeniti onaj: “smeće unutra, ništa van”, “smeće unutra, poruka o pogrešci van” ili “smeću zabranjen ulaz”

❑ **Osnovna pravila kojih se treba držati:**

- Provjeriti ispravnost svih vrijednosti podataka iz vanjskih izvora (datoteka, korisnik, mreža...)
- Provjeriti ispravnost svih vrijednosti ulaznih parametara
- Odlučiti kako postupiti u slučaju neispravnih podataka

Jednostavan primjer defenzivnog programiranja

❑ Nedefenzivno programiranje rekurzivne funkcije faktorijel

```
int faktorijel( int N )
{
    if ( N == 0 ) return 1;
    else return N * faktorijel( N-1 ) ;
}
```

- Za negativne vrijednosti nastupa beskonačna rekurzija.

❑ Defenzivno programiranje rekurzivne funkcije faktorijel

```
int faktorijel( int N )
{
    if ( N <= 0 ) return 1;
    else return N * faktorijel( N-1 ) ;
}
```

- Matematički netočno, ali sprječava beskonačnu rekurziju u slučaju preljeva

Tehnike obrade pogrešaka

❑ Tehnike obrade pogrešaka

- vratiti neutralnu vrijednost (0, "", NULL)
- zamijeniti neispravnu vrijednost sljedećom, moguće ispravnom
 - Npr. while (!read() && !eof())
 - Npr. while (GPSfix != OK) sleep(1/100s) ...
- vratiti vrijednost vraćenu pri prethodnom pozivu npr. ret max(prev, 0)
- zamijeniti neispravnu vrijednost najbližom ispravnom, npr. min(max(kut, 0),360)
- zapisati poruku o pogrešci u datoteku, kombinirano s ostalim tehnikama
- vratiti kôd pogreške (npr. <ERRNO.H>, enum MojGrieh) – pogrešku obrađuje neki drugi dio koda – dojava drugom dijelu:
 - postavljanjem "globalne" varijable statusa
 - extern int errno; char *sys_errlist[] ;
 - preko imena funkcije (negativna vrijednost, 0)
 - bacanjem iznimki (u nastavku)
- pozvati "globalnu" metodu za obradu pogreške, npr. perror()
- bezuvjetni završetak programa, npr. Application.Exit (CancelEventArgs)

❑ Robustnost i ispravnost (programa)

- robustnost - u slučaju pogreške omogućen je daljnji rad programa, iako to ponekad znači vratiti neispravan rezultat
- ispravnost - nikad ne vratiti neispravan rezultat, iako to značilo ne vratiti ništa

Iznimke (Exceptions)

- ❑ Iznimka predstavlja problem ili promjenu stanja koja prekida normalan tijek izvođenja naredbi programa
- ❑ U programskom jeziku C#, iznimka je objekt instanciran iz razreda koji nasljeđuje *System.Exception*

- ❑ **System.Exception – osnovni razred za iznimke**
 - StackTrace – sadrži popis poziva postupaka koji su doveli do pogreške
 - Message – sadrži opis pogreške
 - Source – sadrži ime aplikacije koja je odgovorna za pogrešku
 - TargetSite – sadrži naziv postupka koji je proizveo pogrešku

- ❑ **Iz razreda Exception izvedena su dva razreda**
 - SystemException – bazni razred za iznimke koje generira CLR
 - ApplicationException – bazni razred za iznimke aplikacije

Ugrađene iznimke - `SystemException`

❑ Neke sistemske iznimke izvedene iz razreda `SystemException`

<code>ArrayTypeMismatchException</code>	tip vrijednosti koji se pohranjuje u polje je različit od tipa polja i implicitna konverzija se ne može obaviti
<code>DivideByZeroException</code>	pokušaj dijeljenja s 0
<code>IndexOutOfRangeException</code>	indeks polja je izvan deklarirane veličine polja
<code>InvalidCastException</code>	nedozvoljena konverzija tipa
<code>OutOfMemoryException</code>	nedostatak memorije za alociranje objekta
<code>OverflowException</code>	preljev pri izračunavanju aritmetičkog izraza
<code>NullReferenceException</code>	referenci nije pridružen objekt
<code>StackOverflowException</code>	stog je prepunjen

Obrada iznimki

- ❑ Obrada iznimki sprječava nepredviđeni prekid izvođenja programa
- ❑ Iznimke se obrađuju u kodu za obradu iznimki (exception handler)
 - Obrada pogreške sastoji se razdvajanju kôda u blokove try, catch i finally

```
try {  
    //dio kôda koji može dovesti do iznimke  
}  
catch (Exception1 exOb) {  
    //kôd koji se obavlja u slučaju iznimke tipa Exception1  
}  
catch (Exception2 exOb) {  
    //kôd koji se obavlja u slučaju iznimke tipa Exception2  
}  
...// ostali catch blokovi  
finally {  
    //kôd koji se obavlja nakon izvođenja try, odnosno catch bloka
```

Postupak obrade iznimki

❑ Varijante obrade

- Za jedan try blok može postojati jedan ili više catch blokova koji obrađuju različite vrste pogrešaka.
- Kada dođe do pogreške u try bloku, a postoji više catch blokova, obavlja se onaj catch blok koji obrađuje nastali tip iznimke. Ostali catch blokovi neće se obaviti.
- Ako postoji više catch blokova, posljednji se navodi blok koji obrađuje općenite iznimke (tipa System.Exception)

❑ Primjer: Kodiranje\TryCatch

Generiranje (bacanje) iznimke naredbom throw

❑ Sintaksa

- `throw exceptOb;`
 - `exceptOb` – objekt tipa `Exception` ili iz njega izvedenih razreda
 - `npr. throw new Exception("Nova iznimka");`
- `throw;`
 - prosljeđivanje iznimke na vanjski blok (rethrow)

❑ Primjer, Kodiranje\ExceptionInfo

- bacanje iznimki i prikaz informacije o iznimci

```
throw new SystemException();  
...  
Console.WriteLine( "Izvor: {0}", e.Source );  
Console.WriteLine( "Postupak: {0}", e.TargetSite );  
Console.WriteLine( "Poruka: {0}", e.Message );  
Console.WriteLine( "Trag: {0}", e.StackTrace );
```

❑ Primjer: Kodiranje\Rethrow

- prosljeđivanje uhvaćene iznimke

Kreiranje vlastitih iznimki

❑ Definiranjem razreda izvedenog iz razreda `ApplicationException`

❑ Primjer:  `Kodiranje\CustomException`

■ `catch (Iznimka e)`

```
class Iznimka : ApplicationException
{
    private string val;
    public Iznimka(string str) : base(str)
    {
        val = str;
    }
    public override string Message
    {
        get
        {
            return "Nije neparan " + val;
        }
    }
}
```


Preporuke za korištenje iznimki

- Bacati iznimke samo u stanjima koja su stvarno iznimna
- Izbjegavati prazne blokove za hvatanje iznimki (“catch { }”)
- Izbjegavati hvatanje osnovne iznimke `Exception`,
 - koristiti specifične iznimke, `SystemException` ili `ApplicationException`, znajući što bacaju vlastite knjižnice
- U poruci iznimke uključiti sve informacije o kontekstu nastanka iznimke
- Zapisivati trag bačenih iznimki (log)
- Prosljeđivanje iznimki raditi samo kada želimo specijalizirati iznimku
- Koristiti iznimke za obavijest drugim dijelovima programa o pogreškama koje se ne smiju zanemariti
- Ne bacati iznimke za pogreške koje se mogu obraditi lokalno
- Izbjegavati bacanje iznimki u konstruktorima i destruktorima, osim ako ih na istom mjestu i ne hvatamo
- Razne tipove pogrešaka obrađivati na konzistentan način kroz čitav kod
- Razmotriti izradu centraliziranog sustava za dojavu iznimki u kodu

Primjer centralizirane obrade iznimki

❏ Primjer: Kodiranje\CentraliziraniSustav

```
public static void ReportError(Exception iznimka) {  
    Console.WriteLine("Neocekvana iznimka:  " + iznimka.Message);  
    Console.WriteLine("Mjesto iznimke:");  
    Console.WriteLine(iznimka.StackTrace);  
    Console.WriteLine("\nPritisni tipku...");  
    Console.ReadLine();  
}  
...  
try {  
    Exception ex = new Exception("\nPoruka iznimke...\n");  
    throw ex;  
}  
catch (Exception ANYexception)  
{  
    ReportError(ANYexception);  
}
```

Tvrdnje

❑ Tvrdnja (Assert)

- logička (Boolean) naredba (metoda u C#) kojom se program testira tako da njen uvjet mora biti istinit, a ukoliko nije, program pada
- tvrdnje se koriste za uklanjanje pogrešaka (debugging) i dokumentiranje ispravnog rada programa
- koriste se u fazi kodiranja, naročito u razvoju velikih, kompliciranih programa, te u razvoju programa od kojih se zahtijeva visoka pouzdanost
- omogućavaju programu da sam sebe provjerava tijekom izvođenja
- pišemo ih na mjestima gdje se pogreške ne smiju pojaviti

❑ Prostor imena **System.Diagnostics**, naredba **Debug.Assert**

- Logički izraz za koji se pretpostavlja (tvrdi) da je istinit
- Poruka koja se ispisuje ako izraz nije istinit

❑ Primjer: 📁 Kodiranje\Assert

```
int brzina = 650;  
Debug.Assert((0 <= brzina) && (brzina <= 600),  
             "Neispravna brzina.");
```

Preporuke za korištenje tvrdnji

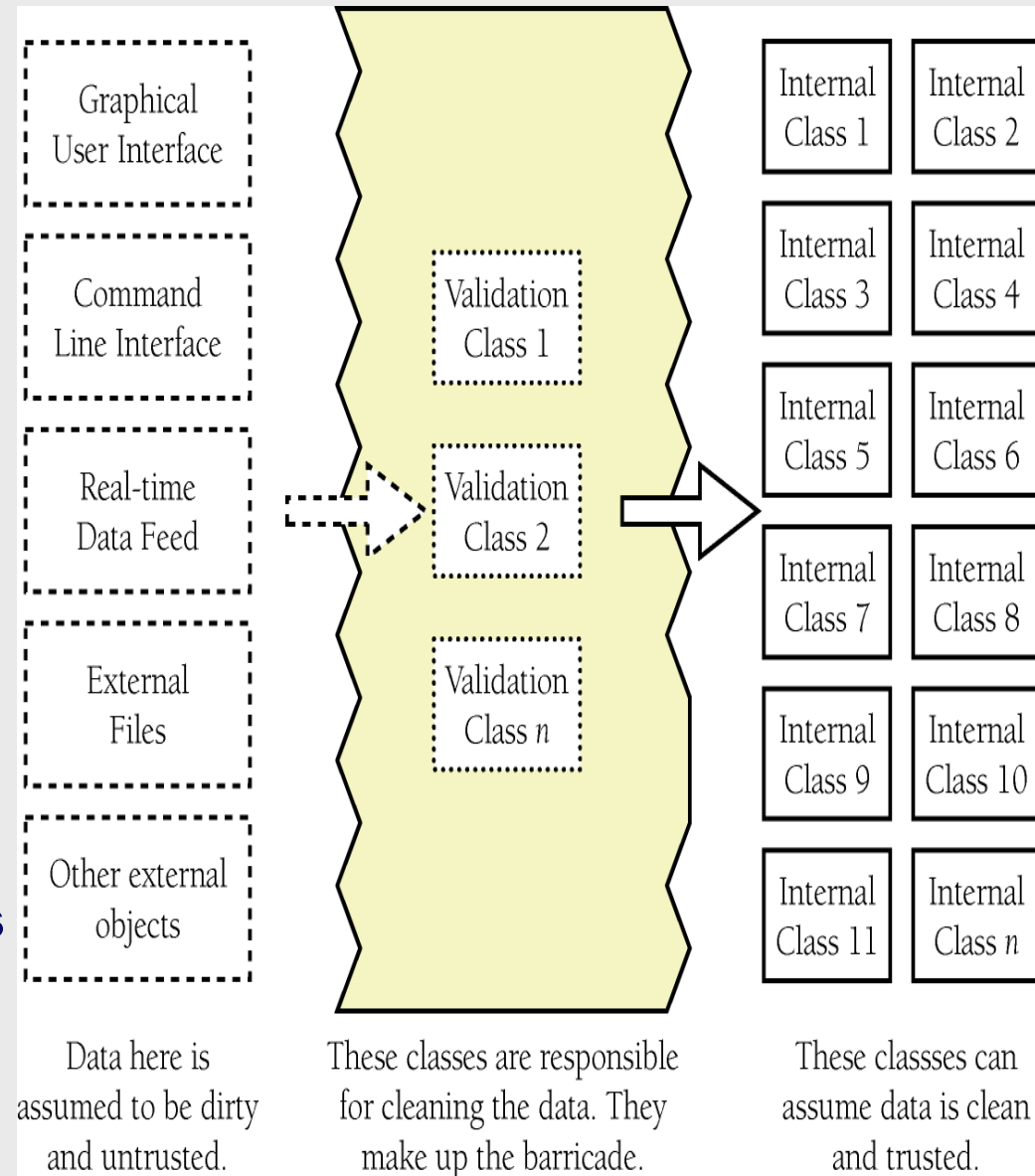
- ❑ Obradu pogreške (iznimke) pisati tamo gdje očekujemo pogreške
- ❑ Koristiti tvrdnje tamo gdje nikad ne očekujemo pogreške
- ❑ Za jako robustan kod koristiti tvrdnje + kod za obradu pogreške

- ❑ Izbjegavati poziv metoda u izrazima tvrdnji
 - `npr. Debug.Assert (Obavi(), "Neobavljeno");`

- ❑ Koristiti tvrdnje
 - za dokumentiranje i verificiranje uvjeta koji moraju vrijediti prije pozivanja metode ili instanciranja razreda (“preconditions”), te
 - uvjeta koji moraju vrijediti poslije djelovanja metode ili rada s razredom (“postconditions”).

Koncept barikada

- ❑ Konstrukcija sučelja kao granica prema “sigurnim” dijelovima koda
- ❑ Definiranje dijelova softvera koji će rukovati “prljavim” (nesigurnim) podacima i drugih koji rukuju samo s “čistim” podacima (validacijskih razreda koji su odgovorni za provjeru ispravnosti podataka i sačinjavaju barikadu, te internih razreda koji rukuju s podacima za koje se pretpostavlja da su provjereni i ispravni)



Koncept barikada

❏ Primjer: Kodiranje\Barikade

```
class Primjer
{
    private int index; //privatna varijabla (iza barikade)
    public Primjer()
    {
    }
    public int Index //javna metoda (sluzi kao sucelje barikada)
    {
        get { return index; }
        set
        {
            //provjeravamo je li unutar zadanih vrijednosti,
            //ako ne, pridjeljujemo najblizu vrijednost
            if (value <= 0) { this.index = 0; }
            else if (value > 100) { this.index = 100; }
            else { this.index = value; }
        }
    }
}
```

Preporuke za korišćenje barikada

- ❑ **Barikade naglašavaju razliku između tvrdnji i obrade iznimaka**
 - Metode s vanjske strane barikade trebaju koristiti kôd za obradu pogreške
 - Unutarnje metode mogu koristiti tvrdnje jer se ovdje pogreške ne očekuju!

- ❑ **Na razini razreda**
 - javne metode rukuju s “prljavim” podacima i “čiste” ih
 - privatne metode rukuju samo s “čistim” podacima.

- ❑ **Pretvarati podatke u ispravan tip odmah pri unosu**

Otkrivanje pogrešaka

- ❑ **Uobičajena zabluda programera je da se ograničenja koja se odnose na konačnu verziju softvera odnose i na razvojnu verziju**
 - Treba biti spreman žrtvovati brzinu i resurse tokom razvoja u zamjenu za olakšani razvoj
- ❑ **Ofenzivno programiranje – učiniti pogreške u fazi razvoja toliko očitim i bolnim da ih je nemoguće zanemariti**
 - osigurati da `assert` naredbe uzrokuju prekid izvođenja pri pogrešci
 - popuniti bilo koju alociranu memoriju prije upotrebe radi detektiranja eventualnih problema s njenom alokacijom
 - popuniti alocirane datoteke ili tokove podataka prije upotrebe radi detektiranja eventualnih grešaka u formatu datoteka ili podataka
 - osigurati da svaka `case` naredba koja propagira do `default` slučaja uzrokuje pogrešku koju nije moguće zanemariti
 - napuniti objekt “smećem” (junk data) neposredno prije njegovog brisanja
- ❑ **Napad je najbolja obrana**
 - pogriješiti toliko jako tijekom razvoja da ne predstavlja problem u pogonu

Otkrivanje pogrešaka (2)

❑ Planirati uklanjanje dijelova programa koji služe kao pomoć u otkrivanju pogrešaka u konačnoj verziji softvera

- koristiti alate za upravljanje verzijama
- koristiti ugrađene predprocesore za uključivanje/isključivanje dijelova koda u pojedinoj verziji
- korištenje vlastitog (samostalno napisanog) predprocesora
- zamjena metoda za otkrivanje pogrešaka u konačnoj verziji “praznim” metodama koje samo vraćaju kontrolu pozivatelju


❑ Primjer: Kodiranje\Debug

```
#define RAZVOJ //definiramo simbol
...
#if (RAZVOJ)
    // kod za debugiranje
    Console.WriteLine("Poruka UNUTAR koda za debugiranje!");
#endif
...
Console.WriteLine("Poruka IZVAN koda za debugiranje!");
```

Količina defenzivnog koda u završnoj verziji

- ☐ **Ostaviti kôd koji radi provjere na opasne pogreške**
- ☐ **Ukloniti kôd koji provjerava pogreške s trivijalnim posljedicama**
 - Ukloniti pretprocesorskim naredbama, a ne fizički
- ☐ **Ukloniti kôd koji može uzrokovati pad programa**
 - U konačnoj verziji treba omogućiti korisnicima da sačuvaju svoj rad prije nego se program sruši.
- ☐ **Ostaviti kôd koji u slučaju pogreške omogućava “elegantno” rušenje programa**
- ☐ **Ostaviti kôd koji zapisuje pogreške koje se događaju pri izvođenju**
 - Zapisivati poruke o pogreškama u datoteku.
- ☐ **Treba biti siguran da su sve poruke o pogreškama koje softver dojavljuje “prijateljske”**
 - Obavijestiti korisnika o “unutarnjoj pogrešci” i navesti e-mail ili broj telefona tako da korisnik ima mogućnost prijaviti pogrešku


Unutarnje iznimke

- ❑ `Exception.InnerException` – dobavlja instancu razreda `Exception` koja je izazvala trenutnu iznimku
- ❑ Primjer:  `Kodiranje\InnerException`

```
class Primjer
{
    public void F()
...
        throw new Exception("Iznimka u Primjer.F() :", e);
..
class Program
{
    static void Main(string[] args)
    {
        catch (Exception e)
        {
            Console.WriteLine("Iznimka u Main: "
                + "{0}\nInnerException: {1}",
                e.Message, e.InnerException.Message);
        }
    }
}
```

Ostale mogućnosti obrade iznimki

- ❑ **C# naredbe mogu se izvoditi u provjerenom (checked) ili neprovjerenom (unchecked) kontekstu.**
 - checked context – aritmetički preljev podiže iznimku
 - unchecked context – aritmetički preljev bude ignoriran, a rezultat "odrezan" (truncated)

- ❑ **Primjer:  Kodiranje\CheckedUnchecked**
 - što se dogodi ako unchecked obavimo bez try ?

```
// upravljanje hvatanjem preljeva  
newVal = checked( op1.Value * op2 ); // uz provjeru  
newVal = unchecked( op1.Value * op2 ); // bez provjere
```

Zadaci za vježbu

- ❑ **Obraditi iznimku kada se ostvari Debug.Assert**

- ❑ **Implementirati razred Niz koji osim niza cijelih brojeva sadržava gornju i donju granicu niza.**
 - Baciti iznimku ako je gornja granica manja od donje, te ako se preko indeksa pristupa članu izvan gornje odnosno donje granice.
 - Napisati vlastiti razred za obradu navedenih iznimki.

- ❑ **Implementirati razred Temperature koji prima razred Niz. Razred treba imati metodu za računanje prosječne temperature.**
 - Ubaciti provjere da niz temperatura nije prazan prije izračuna.
 - Provjeriti i da je dobiveni prosjek unutar granica definirane gornje i donje vrijednosti niza.

Nadzor rada aplikacije

❑ Prostor imena **System.Diagnostics**

- razredi koji omogućavaju ispravljanje pogrešaka (debug) i praćenje izvođenja izvršnog koda (trace)
- `using System.Diagnostics;`

❑ Razred **EventLog** - omogućuje rukovanje dnevnikom događaja

- `WriteEntry` – piše zapis u dnevnik događaja
- `WriteEvent` – piše lokalizirani zapis u dnevnik događaja
- `GetEventLogs` – kreira polje sistemskih događaja

❑ Razred **Debug** - omogućuje ispis na **Output** prozor

- Metode: `Write`, `WriteLine`, `WriteIf`, `WriteLineIf`
- `npr.Debug.WriteLine ("Ispis na Output prozor");`

❑ Razred **Trace** - piše na osluškivač traga (trace listener)

- Metode: `Write`, `WriteLine`, `WriteIf`, `WriteLineIf`

❑ Razred **TraceListener** – osnovni apstraktni razred osluškivača

Reference

- ❑ <http://www.texttrush.com/coding-standard-cs.htm>
- ❑ <http://www.texttrush.com/download/cs-coding-standard-idesign.pdf>

- ❑ **Code Complete (Second Edition), Steve McConnell (poglavlje 8)**
- ❑ www.c-sharpcorner.com
- ❑ www.csharp-station.com