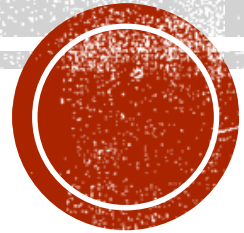
РАЗВОЈ СОФТВЕРА 2

Неке понструкције програмског језика **C#**



КОНСТРУКЦИЈЕ ЈЕЗИКА C#

- Велики број карактеристика и конструкција о програмског језика **C#** је исти или јако сличан као у програмском језику Јава:
  - Начин извршавања (управљаног) програмског кода.
  - Програмске конструкције (гранање, циклуси, дефиниција класе, дефиниција интерфејса, модификатори приступа и сл.).
- Има много различитих елемената:
  - Концепти склопова и простора имена, нове кључне речи, нови програмски конструкти и сл.
- Овде је акценат на опису програмских конструкта који се разликују од оних са којима смо се упознали у програмском језику Јава.
- За сваки од програмских конструкција које су дате у презентацији, дат је и пример програмског кода (понекад и више њих) који то илуструје.
  - Примери се налазе у јавно доступном **Github** репозиторијуму на адреси

    https://github.com/MatfRS2/primeri-predavanja-2021-22

1. НОВОСТИ КОД ФУНКЦИЈА

Промене код дефиниције и позива функција се односе на:

1. Опциони параметри, именовање параметара и аргумената
2. Параметри ref
3. Параметри out
4. Параметри in
5. Методи проширења

1.-1. ОПЦИОНИ ПАРАМЕТРИ

- У дефиницији функције тј. метода (као и конструктора, индексера или делегата) се његови параметри могу специфицрати као обавезни или као опциони.
- Позив функције мора обезбедити све обавезне параметре, али опциони праметри могу да не буду специфицирани - тада се у супституцији користи њихова подразумевана вредност.
- Сваки опциони параметар има подразимевану вредност у оквиру своје дефиниције, па ако вредност аргумента за тај параметар није специфицирана у позиву, користиће се та подразумевана вредност.
- Опциони параметри су дефинисани на карју листе параметара, после свих обавезних.
- Ако се при позиву обезбеђује вредност за неки опциони параметар, потребно је обезбедити вредности за све опционе параметре који му претходе.
- Илустрација опционих параметара је дата у примерима 1 и 2.

1.-1. ИМЕНОВАНИ АРГУМЕНТИ

- Именовани аргументи ослобађају прогамера обавезе уклапања редоследа аргумената при позиву са редоследом параметара у дефиницији метода.
- Информација на који се параметар односи дати аргумент се може дати у позиву метода тако што се специфицира име параметра.
- Именовани аргументи побиљшавају читљивост програмског кода, тако што се лакше идентификује шта представља који аргумент.
- Када се именовани аргументи користе заједно са позиционим аргументима, тада позициони арументи не могу да буду после именованих, и они морају да буду на свом месту.
- Илустрација именованих аргумената је дата у примерима 1 и 2.

1.-2. ПАРАМЕТРИ ref

- Кључна реч ref у листи параметара метода указује да се приликом позива врши супституција по референци, а не по вредности.
- Ова кључна реч чини да (формални) параметар буде само друго име за промењиву која је аргумент у позиву функције.
- Ма која операција над параметром у ствари реализује над аргументом (аргумент мора бити променљива).
- Када се користи ref параметар, тада и параметар у декларацији метода и аргумент у позиву метода морају бити експлицитно значени са ref.
- Аргумент у позиву маркиран са ref мора бити иницијализован пре позива.
- Илустрација ref параметара је дата у примерима 3 и 4.

1.-3. ПАРАМЕТРИ out

- Кључна реч out у листи параметара метода указује да се приликом позива врши супституција по референци.
- Ова кључна реч чини да (формални) параметар буде само друго име за променљиву - аргумент у позиву функције.
- Ма која операција над параметром у ствари реализује над аргументом (аргумент мора бити променљива).
- Личи на ref, само што се овде не захтева да аргумент буде иницијализован пре позива функције.
- Када се користи out параметар, тада и параметар у декларацији метода и аргумент у позиву метода морају бити експлицитно значени са out .
- Илустрација out параметара је дата у примеру 5.

1.-4. ПАРАМЕТРИ in

- Кључна реч **in** у листи параметара метода указује се ради о улазном параметру, па није допуштена његова промена у телу метода.
- Овде се захтева да аргумент буде иницијализован пре позива функције.
- Када се користи **in** параметар, тада и параметар у декларацији метода и аргумент у позиву метода морају бити експлицитно значени са **in**.
- Илустрација **in** параметара је дата у примеру 6.

1.-5. МЕТОДИ ПРОШИРЕЊА

- Методи проширења омогућују да се додтни методи придруже постојећем типу без креирања новог изведног типа, рекомпилацији и без било каквог модификовања оригиналног типа.
- То су статички методи, али такви да се позивају као да су методи примерка типа који се проширује.
- Клијентски код позива методе проширења на потпуно исти начин као што се позивају методи примерка дефинисани у датом типу.
- Илустрација метода проширења је дата у примеру 7.

2. НОВОСТИ КОД КЛАСА

Промене код дефиниције коришћења класа се односе на:

- Склопови и простори имена
- Другачија синтакса код наслеђивања и конструктори
- Другачија правила видљивости за поља/методе и модификатори приступа
- Другачије превазилажење метода
- Парцијалне класе
- Преоптерећење оператора

2.-1. СКЛОПОВИ

- Склоп (енг. **assembly**) је основна јединица за испоруку, контролу верзије, поновну искористивост, опсеге активације и постављање сигурносних права код **.NET** апликација.
- То је скуп типова и ресурса који су преведени и изграђени да раде заједно и да оформе логичку јединицу функционалности
- Склопови имају обик извршне датотеке (**.exe**) или библиотеке за динамичко везивање (**.dll**).
- Укратко, склопви су градивни блокаови за **.NET** апликације.

2.-1. ПРОСТОРИ ИМЕНА

- Простори имена служе за организовање класа.
- То је именовна група класа које имају заједничке карактеристике.
- Они помажу у контроли опсега за методе и класе код већих .NET програмских пројеката.
- Може се рећи да они обезбеђују да један скуп имена (нпр. имена класа) не буде у колизији са другим скупом имена.
- Простори имена могу садржавати класе, просторе имена, интерфејсе, структуре и делегате.
- Укључивање другог простора имена се постиже преко кључне речи **using**.
- Илустрација коришћења простора имена је дата у примеру 0.

2.-2. СИНТАКСА НАСЛЕЂИВАЊА

- Илустрација за другачију синтаксу наслеђивања је дата у примеру 1.

2.-3. МОДИФИКАТОРИ ПРИСТУПА

Модификатор приступа	Значење
public	Нема ограничења приступа.
protected	Приступ је ограничен на класу која садржи елеменат и на типове изведене из те класе (типове који наслеђују ту класу).
internal	Приступ је ограничен на текући склоп.
protected internal	Приступ је ограничен на текући склоп и на и на типове изведене из класе која садржи дати елемент.
private	Приступ је ограничен на тип који садржи дати елемент.
private protected	Приступ је ограничен на класу који садржи дати елемент и на типове у текућем склопу изведене из класе која садржи дати елемент.

2.-3. МОДИФИКАТОРИ ПРИСТУПА

Ниво приступа **public**

- Приступ јавном елементу је допуштен из сваког дела програма.
- То значи да ма који други метод или ма који **d**руги склоп који садржи референцу на **k**ласу може приступити елеме**e**тну означеном са **public.**
- Овај модификатор приступа даје најшире допуштење у поређењу са осталим нивоима приступа.
- Илустрација за ниво приступа **public** је дата у примеру 0.

2.-3. МОДИФИКАТОРИ ПРИСТУПА

Ниво приступа **protected**

- Приступ заштићеном елементу је ограничен на класу која садржи тај елеменат и на класе које њу наслеђују.
- То значи да се у свакој од класа која наслеђује дату класу може приступити заштићеним елементима дате класе.
- Илустрација за ниво приступа **protected** је дата у примеру 2.

2.-3. МОДИФИКАТОРИ ПРИСТУПА

Ниво приступа internal

- Код овог нивоа, приступ је ограничен на склоп у ком се класа налази.
- Илустрација за ниво приступа internal је дата у примеру 3.

Ниво приступа protected internal

- Код овог нивоа, приступ је ограничен на склоп у ком се класа налази и на и на типове изведене из садржајуће класе (који могу бити ван текућег склопа).
- Илустрација за ниво приступа internal је дата у примеру 4.

2.-3. МОДИФИКАТОРИ ПРИСТУПА

Ниво приступа private

- Код овог нивоа, приступ је ограничен на класу која садржи дати елемент.
- Илустрација за ниво приступа internal је дата у примеру 5.

Ниво приступа private protected

- Код овог нивоа, приступ је ограничен на на класу који садржи дати елемент и на типове у текућем склопу изведене из те класе.
- Илустрација за ниво приступа internal је дата у примеру 6.

2.-4. ПРЕВАЗИЛАЖЕЊЕ МЕТОДА

- За разлику од Јаве, где су сви методи подразумевано били виртуални, код језика **C#** су виртуални само они методи који експлицино означени, коришћењем кључне речи **virtual.**
- Метод у подткласи који превазилази виртуални метод такође треба да буде експлицино означен, коришћењем кључне речи **override.**
- Ако треба направити нови метод у поткласи, са истим потписом као метод наткласе, тада тај нови метод треба означити помоћу кључне речи **new.**
- Илустрација за другачије механизме код превазилажења метода је дата у примерима 7 и 8.

2.-5. ПАРЦИЈАЛНЕ КЛАСЕ

- Могуће је раздвојити дедефиницију класе структуре и интерфејса на две или више изворних датотека, које онда буду комбиноване приликом превођења апликације.
- Дељење класе може бити пожељно у следећим ситуацијама:
  - Више програмера у истом великом пројекту ради на истој класи.
  - Ради се са изворним кодом који се аутоматски генерише од стране развојног окружења .
  - Користе се генератори изворног кода којима се додатна функсионалност додаје у класу.
- Дељење класе се постиже коришћењем кључне речи partial.
- Илустрација за парцијалне класе је дата у примеру 9.

2.-6. ПРЕОПТЕРЕЋЕЊЕ ОПЕРАТОРА

- Следећи оператори се могу преоптеретити:

Оператори	Напомене
+x, -x, !x, ~x, ++, --, true, false	Оператори **true** и **false** морају бити заједно преоптерећени.
x + y, x - y, x * y, x / y, x % y, x & y, x | y, x ^ y, x << y, x >> y, x >>> y	
x == y, x != y, x < y, x > y, x <= y, x >= y	Преоптерећивање се мора вршити у паровима: == и !=, < и >, <= и >=.

- Преоптерећење оператора се реализује преко кључне речи operator.
- Илустрација за преоптерећење оператора је дата у примеру 10.

3. НОВОСТИ КОД ИНТЕРФЕЈСА

Промене код дефиниције интерфејса се односе на:

1. Експлицитну имплементацију интерфејса (примери 1 и 2)

4. ОСОБИНЕ

Још један концепт који није постојао у Јави су особине.

1. Поље које подржава особину (примери 1 и 2)
2. Сакривање особина (пример 3)
3. Апстрактне особине (пример 4)
4. Аутоматски имплементиране особине (пример 5)

5. ЕНУМЕРИСАНИ ТИПОВИ

Постоје разлике између енумерисаних типова у Јави и у C#.

1. Дефиниција и конверзија енумерисаних типова (примери 1 и 2)

6. ГЕНЕРИЧКИ ТИПОВИ

Фокус је на разликама између генеричких типова у Јави и у C#.

1. Дефиниција генеричких типова (пример 1)
2. Генеричка листа (пример 2)
3. Генерици и једнакост (пример 3)

7. ИНДЕКСЕРИ

Опис индексера у C#.

1. Дефиниција индексера (пример 1)
2. Преоптерећење индексера (пример 2)
3. Генерички ииндексери (пример 3)

8. ДЕЛЕГАТИ

Опис делегата у C#.

1. Дефиниција делегата (примери 1 и 2)
2. Мултикаст делегати (примери 3 и 4)
3. Генерички делегати (пример 5)
4. Делегати за func, action, predicate (примери 6, 7 и 8)

9. ДОГАЂАЈИ

Опис догађаја у C#.

1. Дефиниција догађаја и руковаоца (примери 1 и 2)
2. Прослеђивање података код догађаја (пример 3)

10. СТРУКТУРЕ

Опис структуре у C#.

1. Дефиниција структуре (пример 1)
2. Структуре, методи и особине (пример 2)
3. Структуре и догађаји (пример 3)

11. ЛАМБДА

Ламбде се користе за креирање анонимне функције, уз помоћ ламбда оператора, тј. =>

Свака ламбда се може претворити у тип делегата (на основу типова својих параметара и типа повратне вредности):

- Ако ламбда не враћа вредност, она се може претворити у **Action** тип делегата.
- Ако ламбда враћа вредност, она се може претворити у **Func** тип делегата.

Када је ламбда облика **(parametri) => izraz** , тада се она назива се ламбда израз.

Када је ламбда облика **(parametri) => {<sekvenca-naredbi>}** , тада се она назива ламбда наредба.

11. ЛАМБДА

1. Дефиниција ламбде (пример 1)
2. Ламбде и догађаји (пример 2)
3. Ламбде и интерфејси (пример 3)
4. Ламбде, особине и иницијализација (примери 4 и 5)
5. Ламбде и делегати (пример 6)
6. Ламбде и индексери (пример 7)

12. ТОРКЕ

Опис торки у C#.

1. Дефиниција торки (пример 1)
2. Угнеждене торке (пример 2)
3. Торке и методи (пример 3)
4. Торке и вредносни типови (примери 4 и 5)
5. Деконструкција торки (пример 6)

13. АСИНХРОНО ПРОГРАМИРАЊЕ

Опис конструкција за асинхроно програмирање у C#.

1. Блокирање (пример 1)
2. Секвенцијални рад(пример 2)
3. Конкурентни рад и задаци (пример 3)
4. Композиција задатака (примери 4, 5 и 6)

14. НУЛАБИЛНИ ТИПОВИ

Опис третирања нулабилности у C#.

1. Дефинисање и коришћење (пример 1)

15. СЛОГОВИ

Опис слогова у C#.

1. Дефинисање слогова (пример 1)
2. Приступ компонентама слогова (примери 2 и 3)
3. Наслеђивање и слогови (пример 4)

16. АТРИБУТИ

Атрибути представљају моћан метод да се мета-подаци декларативно придруже коду (склоповима, типовима, методама, осоабинама итд.). По придруживању атрибута елементу програма, тај атрибут и информације које он носи могу бити испитане помоћу рефлексије.

- Атрибути додају мета-податке у програм. Сви склопови у **.NET** садрже мета-податке који описују типове и њихове елементе унутар склопа. Поред тога, могуће је додати и друге мета-податке.
- Атрибут или више атрибута се могу применити на целе склопове, на модуле, али и на мање елементе као што су класе и особине.
- Атрибути могу имати аргументе исто као што је то случај са методама.
- Коришћењем рефлексије, програмер може испитивати мета-податке који су придружени програму – како оне које је он придружио, тако и све остале.

16. АТРИБУТИ

Неки уобичајени обрасци коришћења атрибута у програмском коду :

- Коришћење атрибута **WebMethod** у веб сервисима да се дати метод може користи преко **SOAP** протокола.
- Описивање како се параметри метода укапају са нативним програмским кодом, помоћу **MarshalAs.**
- Опис **COM** особина за класе, методе и интерфејсе.
- Специфицирање позива неуправљаног (**unmanaged**) кода помоћу атрибута **DllImport.**
- Опис склопа – назива, верзије, описа итд.
- Опис серијализације и опис мапирања класе и **XML** чворова код **XML** серијализације.
- Опис сигурносних захтева за методе.

1. Коришћење атрибута (пример 1)
2. Креирање и процесирање сопствених атрибута (пример 2)

17. LINQ

LINQ тј. упитни интегрисани у језик су скуп технологија којима се интегришу могућности упита у сам **C#** језик.

Традиционални приступ је да упит буде текст који се не проверава током комплације и који зависи од типа извора података.

Захваљујући **LINQ**, код **C#** језика упит је пуноправни грађанин , исто као класе, методи и догађаји.

Надаље, **LINQ** обезбеђује конзистентан запис без обзира на то да ли се упит прави над објектима, релационим базама или **XML**.

Програмери током рада креирају упитне изразе, који се пишу на декларативан начин и који обезбеђују уз минимално кодирање филтерисање, уређење и груписање података.

17. LINQ

LINQ упити се могу писати над **SQL** базама података, над **XML** документима и над ма којом колекцијом објеката која имплементира интерфејсе **IEnumerable** или генерички интерфејс **IEnumerable<T>** .

LINQ се ослања на следеће карактеристике језика **C#**:

- Упитни изрази
- Имплицитно типизиране промењљиве (**var**)
- Иницијализатори за објекте и за колекције
- Анонимни типови
- Методе проширења
- Ламбда изрази

17. LINQ

1. Дефинисање (пример 1)
2. Напредан рад (пример 2)