

**Катедра „Компютърни системи”**

**Специалност „Компютърно и софтуерно инженерство”, степен бакалавър**

Реферат по Програмни среди

**Тема: The dynamic Type**

**Изготвил: Петьо Николаев Цветков**

**Курс III, група 37**

**Факултетен номер: 121217078**

Динамичните типове са въведени в C# 4 (dynamic). Новия тип е стаитичен тип и функционира все едно е от ‘object’ тип. По време на компилиране елементите от този тип се приемат, че поддържат всички опрации. Следователно не трябва да се взима на впредвид от къде се взимат стойностите, дали са от класс String или чрез reflection или някъде другаде от програмата. Ако кода е невалиден то програмата ще се компилира, но по време на работа ще бъдат прихванати грешки. За пример да приемеме, че имаме клас ‘Test’ с една функция в него ‘test’, която приема само един входен параметър. Ако създадем променлива от този тип и извикаме функцията с подаде 2 елемента като параметри, то компилатор няма да позволи да се, защото няма да бъде валидно извикването на функцията с два параметъра. Но ако създадем променлива с динамичен тип, след което й присвоим стойност от тип ‘Test’ и извикаме нейната функция с два параметъра, то това ще бъде компилирано успешно, но ще предизвика грешка по време на работата на програмата.

1. **static** **void** Main(**string**[] args)
2. {
3. Test ec = **new** Test();

6. dynamic dynamic\_ec = **new** Test();
8. dynamic\_ec.test(10, 4);
10. dynamic\_ec.test("some argument", 7);
11. }

Ролята на компилатор в при тези случаи е да събира информацията за всички стъпки, които се изпълняват за обекта от тип ‘dynamic’. По време на работата всички записани операции се изпълняват и ако някоя от тях е невалидна, то тя ще предизвика ‘run-time exception’.

Преобразуването на динамични типо към нединамични става лесно. Това позволява на програмистите лесно да преминава от динамичен към нединамичен контекст. Всеки тип може да бъде преобразувам не динамичен без извършване на допълнителни действия.

1. dynamic d1 = 7;
2. dynamic d2 = "a string";
3. dynamic d3 = System.DateTime.Today;
4. dynamic d4 = System.Diagnostics.Process.GetProcesses();

Преобразуването от динамичен към недимичен тип също може да бъде направено без никакви допълнителни действия, но това може да доведе до ‘run-time exception’.

1. **int** i = d1;
2. **string** str = d2;
3. DateTime dt = d3;
4. System.Diagnostics.Process[] procs = d4;

Динамичния език по време на работа на програмата е въведен в .NET Framework 4. Той предоставя инфраструктур, която поддържа динамични типове в C# и също имплементация на динамичен програмен език като IronPython и IronRuby. Динамичния програмен език може да определи типа по време на работа на програмата, докато преди това в езика C# трябва да посочиш типа по време на създаването на програмата. Като пример за динамични езица са Lisp, Smalltalk, Javascript, PHP, Ruby, Python и Groovy. Провечето от динамичните езици предоставят следните предимства на програмистите:

* Repl(Read-eval-print loop) – това представлява нещо като процес, който ти предоставя да въвеждаш и изпълняваш команди, след което можем да видим резултата на екрана.
* Top-down development (модел на за реализиране на проекти) – това ни предоставя възможноста да извикваме функции от по долни нива преди да сме добавили тяхната имплементация.
* По лесно преправяне на кода – чрез динамичните типове ни предоставя по лесно рефакториране накода, понеже няма да се наложи да се променят използваните типо за функции и променливи

C# 4 включва и допълнителни функционалности заедно с динамичните типове, които ни позволяват по-лесна работа(COM api). Той ни предоставя възмоожноста да наименуваме параметрите на функциите и да ги извикваме по име, а не по ред на подаване.

1. PrintOrderDetails("Gift Shop", 31, "Red Mug");
2. PrintOrderDetails(orderNum: 31, productName: "Red Mug", sellerName: "Gift Shop");

Също така ни предоставя и така наречените не задължителни параметри на фунцкия, които задаваме с стойност по подразбиране, която да използваме ако не сме задали такава като входен параметър.

Кога да използваме динамични типове? Използването на димачини типове ни позволява по – лесна разработка, като не трябва да следим коя променли дали е от същия тип и ни спестява компилационните грешки от типа на несъвпадение на типовете данни. Можем да ги използваме, когато бързодействието на програмата не е от голямо значение, защото по време на работа за нимарането на правилната функция се използва reflection api, който не е много бърз. Също така, когато не ни е от голям приоритет да нямаме run-time exeption-и, след като не е успяно да бъде намерена използваната функция.