[Statically Typed vs Dynamically Typed Languages | Baeldung on Computer Science](https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages" \l ":~:text=Statically%20Typed%20Languages&text=This%20means%20that%20before%20source,Go%2C%20Pascal%2C%20and%20Swift.)

1. \*\*Introduction\*\*

- Brief overview of the topics  
 გამარჯობა მეგობრებო, დღეს ვისაუბრებთ დინამიურ და ანონიმურ ტიპებზე.

- Importance of understanding dynamic types, anonymous objects, and methods

სანამ დეტალურად გავშლით ლექციის თემებს მანამდე ორი სიტყვით ვისაუბროთ იმაზე თუ რატომაა მნიშვნელოვანი დინამიური ტიპების, ანონიმური ობიექტებისა და მეთოდების გამოყენების ცოდნა. უპირველესყოვლისა, ამ feature ების სახელებიდან გამომდინარე, დინამიური, ანონიმური, მიხვდებოდით, რომ ისინი გვეხმარებიან მოქნილობა შევძინოთ ჩვენს ფუნქციონალს. დინამიური ტიპები გვეხმარებიან რომ წინასწარ ტიპიზაციას თავი ავარიდოთ და კომპილატორი, ასე ვთქვათ, მოვატყუოთ, ხოლო ანონიმური ობიექტები და მეთოდები გვეხმარებიან იმაში რომ გზა შევიმოკლოთ, ცალკე კლასები არ ვწეროთ და მარტივი გადაწყვეტილებები შევქმნათ როცა ამის საშუალება გვაქვს.

II. Understanding Events in C# (25 minutes)

Definition and explanation of Events  
[Events - C# Programming Guide | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/)

მანამდე სანამ ივენთებზე საუბარს დავიწყებთ, მოდი გადავიმეოროთ წინა ლექციის მასალა და ვახსენოთ თუ რა არის დელეგატი. დელეგატი არის ერთგვარი პოინტერი (მიმთითებელი) მეთოდზე. იგი წარმოადგენს რეფერენსს მეთოდზე კონკრეტული დასაბრუნებელი ტიპითა და პარამეტრებით. (insert spuderman meme here) ისინი გვადგებიან ხშირად როცა გვჭირდება რომე მეთოდის პარამეტრებში მეთოდი გავგზავნოთ.   
 რაც შეეხება event ებს, ისინი საშუალებას გვაძლევენ შევატყობინოთ სხვა კლასებს, როცა რომელიმე კონკრეტულ კლასში რაიმე „მათთვის საინტერესო“ ხდება. კლასებს რომლებიც აცხადებენ მომხდარი ივენთის შესახებ, ეწოდებათ publisher ები, ხოლო კლასებს რომლებსაც აინტერესებთ მომხდარი ეწოდებათ subscriber ები.   
 event ები მუშაობენ publisher-subscriber მოდელით. Publisher არის ობიექტი, რომელიც შეიცავს event ის განმარტებას, ხოლო subscriber არის ობიექტი რომელიც შეიცავს event ის ჰენდლერს.

A screen shot of a computer program

Description automatically generated

Differentiating Delegates and Events

მთავარი სხვაობა დელეგატებსა და event ებს შორის არის მათ გამოყენების ჩანაფიქრში. Delegate შექმნილია იმისთვის რომ მეთოდის ენკაფსულაცია შევძლოთ, ანუ ჩვენს კლასში იყოს მხოლოდ მეთოდის ხელმოწერა, ხოლო იმპლემენტაცია სადღაც სხვაგან ხელმიუწვდომელ ადგილას, და ასევე მათ შეუძლიათ პირდაპირ გამოიძახონ მეთოდი, რომელსაც წარმოადგენენ (რომლის რეფერენსიცაა მიბმული მათზე) და რომელიც რანთაიმში შეიძლება სხვადასხვა იყოს.   
 ამავდროულად event არის გზა, რომლის დახმარებითაც ერთი კლასი მეორეს ატყობინებს როდესაც მასში მათთვის რამე საინტერესო ხდება. ერთ კლასში არსებულ ივენთს შეუძლია გამოიწვიოს სხვა კლასში არსებული ჰენდლერი მეთდის გაშვება. ანუ კლასში აღძრული event ი არის გზა რომლითაც სხვა კლასებს შეგვიძლია შევატყობინოთ რომ ამ კალსში რაღაც მოხდა და თუ გაინტერსებს რამე ქენი. თუმცაღა განსხვავებით დელეგატებისაგან, გარე კლასებს არ შეუძლიათ პირდაპირ გამოიძახონ event ები ან მიანიჭოს მას პირდაპირ მნიშვნელობა.

Explaining event handlers and event subscribers  
[Publish events that conform to .NET Guidelines - C# Programming Guide | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-publish-events-that-conform-to-net-framework-guidelines)  
[How to subscribe to and unsubscribe from events - C# Programming Guide | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-subscribe-to-and-unsubscribe-from-events)

Event handler ები არიან მეთოდები, რომელთა გამოძახებაც ხდება მაშინ როცა ივენთი ვარდება. ისინი მიენიჭებიან event ს და აქვთ იგივე ხელმოწერა რაც event ს. მაგალითად თუ ჩვენ გვაქვს ივენთი რომელიც წარმოადგენს კლიკს ღილაკზე, ამ ივენთის უკან შეიძლება იყოს არაერთი სახის ჰენდლი ფაილის შენახვით დაწყებული, ბაზაში ჩანაწერის წაშლით დამთავრებული.  
 subscriber ები არიან ობიექტები, რომლებიც მომხდარ event ებს უყურებენ და შემდეგ შესაბამის ჰენდლერს იძახებენ. ისინი ევენთს უსაბსქრაიბდებიან += ოპერატორით.

How to declare, invoke and handle Events

Straight up practice (clock example, shop example)

Practical examples and exercises with Events

Discussing Event-driven programming

2. \*\*Dynamic Types in C# (30 minutes)\*\*

- Introduction to dynamic types and static types (5 min)

როგორც ვიცით, პროგრამირების ენების დაყოფა შეგვიძლია ორ ძირითად კატეგორიად: სტატიკური ენები და დინამიური ენები. მათ შორის მთავარი სხვაობაა თუ როგორ ემუშავებიან ისინი ტიპებს.   
 სტატიკური ენები (statically typed languages) ძირითადად ბრუნავენ სტატიკურად მინიჭებული ცვლადების გარშემო (ანუ პირდაპირ ხელით რომ ვწერთ ტიპს). დინამიურ ენებში ცვლადის ტიპი არაა საჭირო რომ წინასწარ იყოს მითითებული. სხვა სიტყვებით რომ ვთქვათ statically typed ენებში ცვლადების ტიპიზაცია მოწმდება კომპილაციისას ხოლო დინამიურში რანთაიმში.  
მაგრამ რას ნიშნავს ეს?  
 სტატიკურ ენებში ცვლადს ენიჭება მისი მნიშვნელობა როცა პროექტი კომპილირდება და იგი რჩება ამ ტიპში მთელი პროცესის განმავლობაში. Dynamic ენებში ცვლადმა შეიძლება შეიცვალოს ტიპი მრავალჯერ მაშინაც კი როცა აპლიკაცია მუშაობს. (runtime ში)

- Explanation of the dynamic keyword in C# and its use cases (10 min)

Dynamic ტიპი რეალურად სტატიკური ტიპია, მაგრამ dynamic ი გვერდს უვლის სტატიკური ტიპის შემოწმებებს. ძირითად შემთხვევაში, იგი მუშაობს როგორც object ტიპი. კომპილატორი უშვებს რომ dynamic ტიპი ასაპორტებს ყველა ოპერაციას. შესაბამისად ჩვენ არ გვიწევს დავადგინოთ ობიექტი საიდან მიიღებს მნიშვნელობას. თუმცაღა თუ არავალიდურ კოდს დავწერთ შეცდომას შევეჩეხებით runtime ში.   
მაგალითად შევქმნათ კლასი რომელსაც რაიმე მეთოდი აქვს 1 პარამეტრით. შემდეგ ავდგეთ და გავაკეთოთ ამ ობიექტის ინიციალიზაცია, თუმცა დეკლარაციისას ტიპად მივუთითოთ DYNAMIC. შემდეგ ვცადოთ გამოვიძახოთ ეს მეთოდი ჯერ სწორი რაოდენობის პარამეტრებით შემდეგა რასწორი, და ავხსნათ რატომ არ მოგვცა კომპილატორმა შეცდომა.   
  
 მსგავს შემთხვევებში, როცა dynamic ს ვიყენებთ, კომპილატორის როლია, რომ შეაგროვოს ინფორმაცია იმაზე თუ რას ვაკეთებთ dynamic ობიექტსა თუ გამოსახულებაზე. Runtime ამოწმებს შენახულ ინფორმაციას და თუ რომელიმე დებულება არასწორია მაშინ run time ექსეფშენს ვიღებთ. Dynamic ტიპზე ჩატარებული ქმედებების უმეტესობა შედეგად ისევ dynamic ს გვიბრუნებს. ოპერაციები რომლებშიც ისევ dynamic ს არ ვიღებთ არის:  
1) კონვერტაცია dynamic დან  
2) კონსტრუქტორის გამოძახებები, რომელიც dynamic ს შეიცავს.

კონვერტაციები:  
 კონვერტაციები დინამიურ ობიექტებს და სხვა ტიპებს შორის მარტივია. კონვერტაცია საშუალებას გვაძლევს რომ დინამიურიდან არადინამიურ ქცევაზე დავბრუნდეთ და პირიქით. ჩვენ შეგვიძლია ყველა ტიპი გადავიყვანოთ დინამიკში ვნახოთ მაგალითი. და ასევე შეგვიძლია ყველა ტიპი გადმოვიყვანოთ უკან.

- The difference between dynamic and var keywords (5 min)  
 შეამჩნევდით რომ dynamic ის ქცევა ასე ვთქვათ ჰგავს var keyword ის ქცევას. მაგრამ ნუ მოტყუვდებით ეს მხოლოდ ვიზუალურია. Var მიუთითებს იმაზე რომ ცვლადის ტიპი უნდა დაისეტოს კომპილაციის დროს ინიციალიზაციის მიხედვით. ამავდროულად dynamic ნიშავნს რომ ამ ტიპის ცვლადი არის dynamic type. საბოლოო ჯამში dynamic სა და var ს ერთმანეთის საპირისპირო არსი აქვთ. Var გვეხმარება რომ static typing ი უფრო მარტივად გამოვიყენოთ და მისი გამოყენებისკენ გვიბიძგებს. იგი უზურველყოფს რომ ცვლადს ჰქონდეს შესაბამისი ტიპი, რომელიც ინიციალიზერმა დააბრუნა.  
 dynamic keyword ი სტატიკური ტიპიზაციის თავიდან არიდებისთვის გამოიყენება. როდესაც ცვლადის დეკლარაციაში dynamic ს ვიყენებთ იგი კომპილატორს ანიშნებს რომ მან საერთოდ შეწყვიტოს ტიპის დადგენაზე მუშაობა. Dynamic ცვლადი მიიღებს ტიპს, რომელიც მას runtime ში მიენიჭება.   
 კიდევ ერთი სხვაობა ამ ორ ქივორდს შორის არის ის რომ var გვხვდება მხოლოდ ლოკალური ცვლადის დეკლარაციაში, ჩვენ არ შეგვიძლია var გამოვიყენოთ property ის ტიპის მისათითებლად და არც დასაბრუნებელ პარამეტრად შეგვიძლია მისი მითითება.  
 როგორც დეველოპერებმა შეიძლება გამოვიყენოთ dynamic მაშინ როცა არ ვიცით რა ტიპი დაგვიბრუნდება სხვა კოდიდან (მეთოდიდან/API call დან) ან არ ვიცით რას დაგვიბრუნებს DOM (document object model).

- Advantages and disadvantages of using dynamic types (5 min)  
 სტატიკური ენები ზოგადად უფრო სწრაფებად მოიაზრებიან, ვინაიდან ისინი არკვევენ თავის ტიპებს კომპილაციისას, რაც ეხმარება მათ აპლიკაციის პერფორმანსის გაუმჯობესებასა და ოპტიმიზაციაში. Machine language ს და Assembly ის გარდა ყველაზე სწრაფი ენებია C და C++. მათ სისწრაფეს ვერც ერთი სხვა ენა ვერ უტოლდება ვინაიდან მათ აქვთ ყველაზე კარგი მეხსიერების მენეჯმენტი.   
 სისწრაფესთან ერთად სტატიკურ პროგრამირების ენებს ახასიათებთ შემდეგი რამ. ისინი ადვილად ფუჭდებიან და ამას მალევე ვიგებთ კომპილაციისას, მანამდე სანამ კოდს გავუშვებთ საერთოდ. რაც ნიშნავს იმას რომ ბერად ნაკლები ბაგი გვექნება რანთაიმში და შესაბამისად ლაივზე.   
 ახლა შევხედოთ იგივე კუთხით dynamic ენებს ან ისეთ ენებს სადაც dynamic გამოიყენება. უპირველეს ყოვლისა Dynamic ძალიან მარტივს ხდის კოდის წერის პროცესს, ვინაიდან არ გვიწევს ტიპებზე ზრუნვა და კომპილატორი არ გვეჩხუბება ყოველ მეორე წამს. თუმცაღა Dynamic პროცესი ვინაიდან რანთაიმში ადგენს ტიპებს ბევრად უფრო ნელია. ასევე ჩვენ შეიძლება შევცვალოთ მარტივი რამ, რომელზეც მთელი ლოგიკაა დაფუძვნებული და ვერ გავიგოთ რომ ბაგი გამოვიწვიეთ მანამდე სანამ ლაივზე არ შევეჩეხებით პრობლემას.

- Detailed walk-through of coding examples using dynamic types (5 min)

3. \*\*Break (5 minutes)\*\*

4. \*\*Anonymous Types in C# (30 minutes)\*\*

- Introduction to anonymous types in C# (5 min)  
 ანონიმური ტიპები გვაწვდიან საშუალებას რომ რამდენიმე read only ფროფერთი გადავაკეთოთ ერთიან ობიექტად მარტივად, ცალკე ტიპის შექმნის გარეშე. ტიპის სახელი გენერირდება კომპილატორის მიერ და კოდის დონეზე ხელმისაწვდომი არაა. ანონიმური ტიპებისთვის property ებისთვის ტიპის განსაზღვრა კომპილაციისას ხდება.   
 ანონიმური ტიპების შექმნა შეგვიძლია new ოპერატორისა და object initializer ის კომბინაციით. ვნახოთ მარტივი მაგალითი.  
 ანონიმური ტიპები ძირითადად გამოიყენება select ში როცა გვაქვს რაღაც რაღაც სელექტის ქუერი და კონკრეტული პარამეტრების დაბრუნება გვინდა უკან.   
ანონიმური ტიპები შეიცავენ ერთ ან მეტ public property ის. ანონიმურ ტიპებში სხვა წევრები არაა დაშვებული (მეთოდები, ფილდები...). გამოსახულები რომელიც ანონიმური ტიპის ფროფერთის ინიციალიზაციისთვის გამოიყენება არ შეიძლება იყოს null, ანონიმური ფუნქცია ან pointer ტიპი.   
 ანონიმური ტიპის გამოყენების ერთ-ერთი ყველაზე ხშირი სცენარია ანონიმური ტიპის შექმნა სხვა ტიპის property ებისაგან. განვიხილოთ მაგალითზე. მაგალითად გვაქვს ტიპი Person, შევქმნათ ამ პერსონების ლისთი და შემდეგ ამ ლისთს ჩამოვურბინოთ ციკლით, ასევე ციკლს გარეთ შევქმნათ dynamic ლისთი. შემდეგ ციკლში ანონიმურ ტიპში დავასელექტოთ პერსონ ობიექტიდან რამდენიმე property და შევინახოთ დინამიურ ლისთში, შემდეგ ასევე ციკლით ჩამოვურბინოთ დინამიურ ლისთს.  
 სტანდარტულად როდესაც ჩვენ ვქმნით ანონიმური ტიპის ცვლადს, ტიპს არ ვწერთ, არამედ ვიყენებთ var ქივორდს. ტიპის სახელს ვერ მივუთითებთ, ვინაიდან ანონიმური ტიპის სახელებზე წვდომა აქვს მხოლოდ კომპილატორს.   
 ჩვენ შეგვიძლია შევქმნათ ანონიმური ტიპების მასივი თუ გავაკეთებთ implicitly typed მასივს და მასში ჩავყრით ასევე implicitly typed ანონიმურ ტიპებს.

Anonymous types are [class](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) types that derive directly from [object](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types), and that cannot be cast to any type except [object](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types). The compiler provides a name for each anonymous type, although your application cannot access it. From the perspective of the common language runtime, an anonymous type is no different from any other reference type.

If two or more anonymous object initializers in an assembly specify a sequence of properties that are in the same order and that have the same names and types, the compiler treats the objects as instances of the same type. They share the same compiler-generated type information.

Anonymous types support non-destructive mutation in the form of [with expressions](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/with-expression). This enables you to create a new instance of an anonymous type where one or more properties have new values:

var apple = new { Item = "apples", Price = 1.35 };

var onSale = apple with { Price = 0.79 };

Console.WriteLine(apple);

Console.WriteLine(onSale);

You cannot declare a field, a property, an event, or the return type of a method as having an anonymous type. Similarly, you cannot declare a formal parameter of a method, property, constructor, or indexer as having an anonymous type. To pass an anonymous type, or a collection that contains anonymous types, as an argument to a method, you can declare the parameter as type object. However, using object for anonymous types defeats the purpose of strong typing. If you must store query results or pass them outside the method boundary, consider using an ordinary named struct or class instead of an anonymous type.

Because the Equals and GetHashCode methods on anonymous types are defined in terms of the Equals and GetHashCode methods of the properties, two instances of the same anonymous type are equal only if all their properties are equal.

Anonymous types do override the ToString method, concatenating the name and ToString output of every property surrounded by curly braces.

6. \*\*Break (5 minutes)\*\*

7. \*\*Anonymous Methods in C# (20 minutes)\*\*

ანონიმური მეთოდი არის მეთოდი სახელის გარეშე. ანონიმური მეთოდები შეგვიძლია შევქმნათ delegate keyword ის დახმარებით და შეეგვიძლია მივანიჭოთ delegate ტიპის ცვლადებს. ანონიმური მეთოდ შეუძლია:  
1)ჰქონდეს წვდომა გარე ცვლადებსა და ფუნქციებაზე  
2)შეგვიძლია გადავაწოდოთ როგორც პარამეტრი  
3)შეგვიძლია გამოვიყენოთ როგორც event handler

Delegate ზე პირდაპირ მეთოდის სახელით მიბმას ეწოდება სახელიანი მეთოდი. დელეგატები შეიძლება გვქონდეს როგორც სახელიანი ასევე უსახელო. მაშინ როდესაც ახალი მეთოდის შექმნა ცალკე თავისტკივილია შეგვიძლია გამოვიყენოთ უსახელო დელეგატი და ადგილზევე შევქმნათ მეთოდის იმპლემენტაცია. შეიძლება ასევე დელეგატი ანონიმური მეთოდის ადგილას გვქონდეს lambda ექსფრეშენი.

ვნახოთ პრაქტიკული მაგალითები

- Definition of anonymous methods and their use cases (5 min)

- Comparison between named methods and anonymous methods (5 min)

- Explanation of the delegate keyword in relation to anonymous methods (5 min)

- Detailed walk-through of coding examples using anonymous methods (5 min)

10. \*\*Summary and Conclusion (15 minutes)\*\*

- Recap of the main points of the lecture

- Discussion of the importance of these concepts in modern programming

- Q&A and feedback session