MÜHENDİSLİK FAKÜLTESİ

YAZILIM MÜHENDİSLİĞİ YÜKSEK LİSANS

TEZSİZ YÜKSEK LİSANS DÖNEM PROJESİ

C# VE JAVA DİLLERİNDE ASYNC PROGRAMLAMANIN PERFORMANSA OLAN ETKİSİNİN İNCELENMESİ

HAZIRLAYAN
Türhan Yıldırım

DANIŞMAN ÖĞRETİM ÜYESİ
Dr. Öğr. Üyesi Bilgin Avenoğlu

2022

ETİK İLKELERE UYGUNLUK BEYANI

Dönem proje yazma sürecinde bilimsel ve etik ilkelere uyduğumu, yararlandığım tüm kaynakları kaynak gösterme ilkelerine uygun olarak kaynakçada belirttiğimi ve bu bölümler dışındaki tüm ifadelerin şahsıma ait olduğunu beyan ederim.

İmza

Adı Soyadı

C# VE JAVA DİLLERİNDE ASYNC PROGRAMALAMANIN PERFORMANSA OLAN ETKİSİNİN İNCELENMESİ

Türhan Yıldırım

AHMET YESEVİ ÜNİVERSİTESİ

YAZILIM MÜHENDİSLİĞİ YÜKSEK LİSANS BÖLÜMÜ

2022

ÖZET

Bu tez kapsamında asenkron uygulama geliştirme C# ve Java dillerinde performansa olan etkisi incelenmektedir. Performans inceleme durumu işlemci ağırlıklı ve dosya okuma işlemleri için olan metodlar için yapılmaktadır.

C# dili için elde edilen sonuçlara göre asenkron programlama işlemci yükü gerektiren uygulamalarda 10 kata yakın performans artışı sağlamıştır. Dosya okuma yazma yükü uygulamalarında ise herhangi bir performans kazancı sağlanamadığı tespit edilmiştir.

Java dili için elde edilen sonuçlar incelendiğinde dosya okuma yazma işlemleri için sonuçlar c# dili ile aynı doğrultuda olduğu görülmüştür. Performans anlamında bir kazanç elde edilmemiştir. Fakat işlemci yükü gerektiren işlemlerin sonuçlar incelendiğinde asenkron kısmın 6 kat daha yavaş çalıştığı tespit edilmektedir. Java dilinde elde edilen bu yavaş çalışma durumu Runnable, FutureTask, ExecutorService asenkron implementasyonu ile geliştirilen işlemler için olduğu unutulmamalıdır.

Anahtar Kelimeler: Sync, Async, Programalama, Performans

Danışman: Dr. Öğr. Üyesi Bilgin Avenoğlu

INVESTIGATION OF THE IMPACT OF ASYNC PROGRAMMING ON PERFORMANCE IN C# AND JAVA

Türhan Yıldırım

AHMET YESEVI UNIVERSITY

SOFTWARE ENGINEERING MASTER'S DEPARTMENT

2022

ABSTRACT

In this thesis, the effect of asynchronous application development on performance in C# and Java languages is examined. Performance review is done for methods that are processorheavy and for file reading operations.

According to the results obtained for the C# language, asynchronous programming has provided a performance increase of nearly 10 times in applications that require processor load. It has been determined that no performance gain can be achieved in file read-write load applications.

When the results obtained for the Java language are examined, it is seen that the results for the file read and write operations are in the same direction as the c# language. No gain was made in terms of performance. However, when the results of the processes that require processor load are examined, it is determined that the asynchronous part works 6 times slower. It should be noted that this slow running state obtained in Java language is for operations developed with asynchronous implementation of Runnable, FutureTask, ExecutorService.

Keywords: Sync, Async, Programming, Performance

Advisor: Dr. Öğr. Üyesi Bilgin Avenoğlu

İÇİNDEKİLER

ETİK İLKELERE UYGUNLUK BEYANI ii
ÖZET iii
ABSTRACT iv
İÇİNDEKİLER v
ŞEKİLLER LİSTESİ vi
TABLOLAR LİSTESİ vi
SİMGELER VE KISALTMALAR vi
BÖLÜM I GİRİŞ 1
1.1. Problem 1
1.2. Araştırmanın Amacı 1
1.3. Araştırmanın Önemi 1
1.4. Sayıltılar 1
1.5. Sınırlılıklar 1
1.6 Tanımlar 1
BÖLÜM II KAVRAMSAL ÇERÇEVE 2
2.1. Literatür Taraması 2
BÖLÜM III YÖNTEM 3
3.1. Araştırmanın Modeli 3
3.2. Evren ve Örneklem 3
3.3. Veri Toplama Araçları 3
3.4. Verilerin Toplanması 4
3.5. Verilerin Analizi 5
Tablo 2. Java Uygulaması için Veriler 6
BÖLÜM IV BULGULAR VE YORUM 6
4.1. Araştırma sorusuna(alt problem) ilişkin bulgular aşağıda verilmiştir. 6
BÖLÜM V SONUÇ, TARTIŞMA VE ÖNERİLER 7
5.1. Sonuç 7
5.2. Tartışma 8
5.3. Öneriler 8
KAYNAKÇA 9
EKLER 10
C# Uygulama Kodları 10
Java Uygulama Kodları 13

ŞEKİLLER LİSTESİ

Şekil 1. C# Senkron (Sync) Akış Uygulama Kodları……10
Şekil 2. C# Asenkron (ASync) Akış Uygulama Kodları……10
Şekil 3. C# Ana İşlem Methodlarının Kodları……11
Şekil 4. C# Örnek Test Çıktısı……11
Şekil 5. Java Senkron (Sync) Akış Uygulama Kodları……12
Şekil 6. Java Asenkron (Async) Akış Uygulama Kodları……13
Şekil 7. Java Ana İşlem Methodlarının Kodları……14
Şekil 8. Java Örnek Test Çıktısı……15

TABLOLAR LİSTESİ

Tablo.1 C# Uygulaması için Verileri……5
Tablo 2. Java Uygulaması için Veriler……6

SİMGELER VE KISALTMALAR

Async: Asenkron

Sync: Senkron

CPU: İşlem Gücü

IO: Input Output

File IO: Dosya Okuma İşlemi

MS: Mili Saniye

NS: Nano Saniye

CPU: İşlemci (Central Processing Unit)

BÖLÜM I
GİRİŞ

1.1. Problem

Günümüz trendi olan asenkron(async) yazılım geliştirme mantığının popüler yazılım geliştirme dillerinden olan C# ve Java'da işlemci yükü gerektiren ve dosya okuma-yazma (File IO) işlemlerine olan etkisi gerçekten ifade edildiği gibi performans artışı sağlamakta mıdır?

1.2. Araştırmanın Amacı

Asenkron programlama ile vaat edilen performans artışı işlem türüne göre farklılık gösteriyor mu tespit etmek istiyoruz. İşlem gücü (CPU) gerektiren ve dosya okuma-yazma (File IO) işlemi yapan işlemler karşılaştırılacaktır. Buna göre bu 2 koşul için performans etkileri incelenecektir.

1.3. Araştırmanın Önemi

Gerçek bir performans artışı sağlanabilecek mi bilinmeden geliştirilen yazılım ilk günden asenkron(async) olarak yazılmaya çalışılmakta ve bu durum geliştirme sürecin de yazılım geliştirme sürenin, kompleksitesinin artmasına sebep olmaktadır.

1.4. Sayıltılar

Herhangi bir sayıltı bulunmamaktadır.

1.5. Sınırlılıklar

İşlemci yükü oluşturan ve dosya okuma-yazma işlemleri için asenkron yazılım geliştirme metodolojisini kullanan tüm yazılımlar için geçerli bir durumdur.

1.6 Tanımlar

Aynı işlemi defalarca kez tekrar eden ya da çok fazla işlem yükü olan uygulamalar/yazlımlar için bu durum geçerlidir. Tek kullanıcısı olan, zaman sınırı olmayan yani işlemin ne kadar süre de ne kadar işlemci yükü ile yapıldığının önemi olmayan yazılımlar için geçerli değildir.

BÖLÜM II
KAVRAMSAL ÇERÇEVE

2.1. Literatür Taraması

Yapılan literatür taramalarına göre asenkron yazılım geliştirme yaklaşımı üzerine oldukça fazla yazı/makale bulunmasına rağmen, var olan araştırmalar asenkron yazılım geliştirme yaklaşımının içsel çalışma mekanizmaları ve algoritmaları üzerinde yoğunlaşmaktadır. Ayrıca doğru bir asenkron yazılım parçasının nasıl yazılması gerektiği üzerinde durmaktadır. Asenkron olarak geliştirilen uygulamaların performans etkisinin analizine odaklanan bir araştırma bulunmamaktadır.

Genel literatür asenkron programlama modelleri ve uygulanışları üzerine yoğunlaşırken, aşağıda belirtilen araştırmada doğru bir şekilde geliştirilen asenkron uygulamalar için %30,7 gibi çok ciddi bir performans kazancı olabileceğini bildirmektedir. Tabi ki bu performans artışını diğer yayımlarda olduğu gibi işlemci yükü gerektiren ya da dosya okuma-yazma işlemi için olan asenkron uygulamalar özelinde araştırılmadığını asenkron uygulama geliştirmeye uygun bir senaryo için yapılan doğru bir asenkron yazılım geliştirme metodolojisinin uygulanması sonucu elde edilen performans değişiminin sonucu olduğunu bilmek gerekmektedir. (Castillo, 2019)

Asenkron işlem mantığına istinaden yayınlanan başka bir yayımda ise, asenkron programlama algoritmalarının senkronize işlem algoritmalarına göre 1,5-12 kat arasında performans kazancı sağladığını göstermektedir. İlgili yayımda FSSP adı verilen asenkron işlem gerçekleştirme algoritmasının sonuçlarını göstermektedir. İlgili yayımda işlemci gücü gerektiren yazılımlar ile dosya okuma-yazma üzerinde yoğunlaşan uygulamalar ayrımı yapılmaksınız performans yorumu yapılmaktadır. (Shi & Chang, 2019)

Java ve X10 yazılım geliştirme dillerini hedef alıp bu dillerde geliştirilen asenkron uygulamalarda oluşabilecek “race condition” yani aynı anda erişilmeye çalışılan bir kaynağın olması ve kaynağın aslında aynı anda sadece bir çağrı cevap verebilmesi sonucu diğer çağrıların kilitlenip sırada kalması ve dar boğaz yaratması durumudur. Bu durum yani makalede anlatılan sonuç aslında bu tez kapsamında ölçemeye çalışılan bir dosyanın okunup-yazılması (IO) işlemine atıfta bulunmaktadır. Çünkü bir metin dosyasının içeriğinin okunma sürecince dosya yazılımsal olarak açılıp ve ilgili dosya içeriği satır satır okunur. Bu okuma işlemi yazılımsal olarak aynı anda sadece bir tane okuma istediğine cevap verebileceği için aslında bu test senaryosunda race condition sebep olması muhtemeldir. İlgili yayımda da bu tarz durumlar da asenkron olarak geliştirilen uygulamanın performans artışı yaşaması yerinde yaklaşık olarak 3.05 kat kadar uygulamanın senkron uygulamaya göre yavaş çalıştığı tespit edilmektedir.(Raman & Yahav, 2010)

BÖLÜM III
YÖNTEM

3.1. Araştırmanın Modeli

Çalışmanın yapılacağı bilgisayar bilgileri aşağıdaki gibidir.

- AMD Ryzen 3900X (12 Fiziksel Çekirdek) İşlemci
- 32 GB 3200MHZ DDR4 Ram
- Samsung 970Evo Plus 512GB SSD Sabit Disk
- Windows 11 (21H2 – OS Build 22000.613) İşletim sistemi

C# için geliştirilecek uygulama ile ilgili bilgiler aşağıda verilmiştir.

- Visual Studio 2022 (17.1.5)
- .NET 6.0
- .NET SDK Version 6.0.202
- Console uygulaması yazılacaktır.
- Asenkron kısmında ilgili methodlar Task.Run şeklinde çağrılacaktır.

Java uygulaması için kullanılacak uygulama bilgileri aşağıda verilmiştir.

- IntelliJ IDEA 2022.1.1 (Community Edition) Build #IC-221.5591.52
- jre1.8.0_331
- Console uygulaması yazılacaktır.
- Asenkron kısımda ilgili methodlar için Runnable task şeklinde çağrılacaktır.

3.2. Evren ve Örneklem

Veri toplama methodlarından elde edilecek sonuçlar kullanılacaktır.

3.3. Veri Toplama Araçları

Verinin toplanması için 2 temel method yazılacaktır.

- İşlemci yükü gereken method, içeriğinde 1’den 10000’e kadar olan sayıların karelerini hesaplanacaktır.
- Dosya okuma işlemi yapacak olan method, içeriğinde 1MB metinsel veri içeren (txt) dosyasını okunacaktır.

Bu iki temel method için C# ve Java dillerinde, ilgili methodları 1000'er kez senkron ve asenkron çağıran methodlar hazırlanacak. Bu işlem her bir durum için 10'ar kez tekrarlanacaktır.

- Senkron bir şekilde 1'den 10000'e kadar olan sayıların karelerini hesaplayan method 1000 kez çağrılacak ve bu akış 10 kez tekrar edilecektir, işlemin kaç milisaniyede tamamlandığı alınacaktır.
- Senkron bir şekilde 1MB metinsel veri içeren txt dosya okunacak 1000 kez çağrılacak ve bu akış 10 kez tekrar edilecektir, işlemin kaç milisaniyede tamamlandığı alınacaktır.
- Asenkron bir şekilde 1'den 10000'e kadar olan sayıların karelerini hesaplayan method 1000 kez çağrılacak ve bu akış 10 kez tekrar edilecektir, işlemin kaç milisaniyede tamamlandığı alınacaktır.
- Asenkron bir şekilde 1MB metinsel veri içeren txt dosya okunacak 1000 kez çağrılacak ve bu akış 10 kez tekrar edilecektir, işlemin kaç milisaniyede tamamlandığı alınacaktır.
- Geliştirilen uygulamalara aşağıdaki adreslerden ulaşabilirsiniz
  - C# Uygulaması: https://github.com/turhany/ThesisApps/tree/main/CSharp
  - Java Uygulaması: https://github.com/turhany/ThesisApps/tree/main/Java

3.4. Verilerin Toplanması

C# ve Java için geliştirilen uygulamalar tamamlanıp çalıştırılmış ve veri analizi bölümündeki çıktılar elde edilmiştir.

3.5. Verilerin Analizi

C# uygulaması için elde edilen veriler aşağıda verilmiştir.

	Dosya Okuma (milisaniye)		**Hesaplama (milisaniye)**	
Deneme	Senkron	Asenkron	Senkron	Asenkron
1	1280	1280	239	113
2	1058	1421	237	14
3	1046	1420	237	15
4	1048	1406	237	14
5	1045	1398	237	14
6	1046	1442	238	14
7	1043	1385	237	14
8	1387	1397	237	16
9	1575	1384	237	15
10	1566	1436	238	15
Ortalama	1209,4	1396,6	237,4	24,4

Tablo.1 C# Uygulaması için Verileri

Java uygulaması için elde edilen veriler aşağıdadır.

Hesaplama işlemlerinin yapıldığı methodun asenkron halinde işlemler çok hızlı tamamlandığında milisaniye olarak sonuçlar sıfır(0) olarak ölçülmektedir. Bu yüzden işlem yapıldığını ifade edebilmek için hesaplama işleminin senkron hesaplamaları için nano saniye bilgisi de verilmiştir.

	Dosya Okuma (milisaniye)		**Hesaplama (milisaniye)**	
Deneme	Senkron	Asenkron	Senkron	Asenkron
1	1080	811	1 (1586100 ns)	36
2	828	804	0 (22200 ns)	3
3	651	864	0 (20900 ns)	2
4	646	693	0 (21800 ns)	2
5	645	834	0 (22000 ns)	2
6	644	631	0 (22300 ns)	3
7	642	703	0 (20100 ns)	2
8	648	636	0 (22300 ns)	1
9	643	664	0 (22000 ns)	1
10	644	643	0 (22100 ns)	1
Ortalama	707	728	0 (178180 ns)	5

Tablo 2. Java Uygulaması için Veriler

BÖLÜM IV

BULGULAR VE YORUM

4.1. Araştırma sorusuna(alt problem) ilişkin bulgular aşağıda verilmiştir.

Hazırlanılan C# ve Java uygulamasının çıktıları incelendiğinde aşağıdaki sonuçlara ulaşılmıştır.

C# uygulama özelinde incelendiğinde hesaplama methodu senkron işlem akışında görevini ortalama olarak 238.1 milisaniyede tamamlamıştır. Aynı işlem asenkron akış ile gerçekleştirildiğinde ise tüm işlemler ortalama 24.8 milisaniye sürmüştür. Buda c# uygulamasının hesaplama adımı için asenkron programlamanın yaklaşık 9.8 kat daha hızlı çalışmasını sağlamıştır. Elde edilen bu bilgiye göre işlemci yükü gerektiren işlemler için c# yazılım dilinde asenkron programlamanın performans artışı sağladığı saptanmıştır.

Yine C# uygulamasının dosya okuma akışı verileri incelendiğinde senkron akış ortalama 1236.1 milisaniye sürerken asenkron akış ise ortalama 1385 milisaniye sürmüştür. Bu verilere göre

ise dosya işlemleri için asenkron akışın uygulanmasının performans artışına bir etkisinin olmadığı gibi yaklaşık %12 gibi bir performans kaybına da sebep olmuştur.

Java uygulamasının verileri incelendiğinde ise dosya işlemleri için C# uygulaması çıktıları ile aynı durum görülürken, hesaplama işlemi için ise tersine bir durum oluştuğu görülmektedir.

Java uygulamasının dosya okuma kısmı için senkron akış ortalama olarak 710 milisaniye sürmektedir. Asenkron akış ise 731 milisaniye sürmektedir. Bu Java yazılım dilinde asenkron dosya okuma işlemi yapan uygulamalar için senkron akışa göre yaklaşım %2,9 oranında bir yavaşlama demek olmaktadır.

Geliştirilen Java test uygulamasının işlemciye yüklenen hesaplama kısmı için ise senkron akışın ortalaması nano saniye(178180 ns – 0.17ms) seviyelerinde olurken senkron akışın ortalaması ise 6 milisaniye olduğu görülmektedir. Buna göre geliştirilen asenkron uygulaması senkron uygulamaya göre yaklaşık 6 kat yavaş olduğu tespit edilmiştir.

BÖLÜM V
SONUÇ, TARTIŞMA VE ÖNERİLER

5.1. Sonuç

Toplanan test verilene göre c# ve Java uygulamalarında dosya okuma yazma işlemleri için asenkron programlama performans kazancı sağlamamakla birlikte uygulamaların daha yavaş çalışmasına sebep olmaktadır. Elde edilen sonuçlara göre denilebilir ki eğer aynı anda sadece bir işleme cevap veren bir kaynak üzerinde asenkron yazılım geliştirmesi herhangi bir performans kazancı sağlayamayacaktır. Çünkü ilgili kaynak ki test senaryosunda dosya okuma işlemi olmakta ve bir dosya aynı anda sadece bir işlem yapılmasına izin verdiğinde asenkron geliştirilen kod blokları da kendi aralarında sıraya girip tek tek işlem yapmaktadır. Aktif olarak sadece bir işlem olabilmekte diğer işlemler bloklanmaktadır.

İşlemci gücüne dayanan hesaplama methodun da ise c# için elde edilen test sonuçlarına göre 9.8 kat performans artışı sağlandığı görülmektedir. Bu durumda sonuçlara göre net bir şekilde tespit edilmektedir ki asenkron programlama işlem gücü gerektiren işlemler için performans artışı sağlamaktadır. Buradaki asıl etki işlem gücü gerektiren işlemlerin, dosya okuma gibi bloklanmaması ve ayrı ayrı task'lar içinde asenkron olarak işlenebilmesidir.

Fakat Java yazılım dili ile elde edilen sonuçlar incelendiğinde hesaplama işlemleri içeren Java uygulamaları için aynı sonuç ortaya çıkmamıştır. Asenkron şeklinde yazılan işlemler 6 kattan fazla yavaş çalışmaktadır. Java yazılım dilinde "ExecutorService" ile yazılan işlem gücüne dayalı asenkron uygulamalar için performans artışı sağlanamadığı tespit edilmiştir.

5.2. Tartışma

Yapılan asenkron programlama testlerinin sonucuna göre c# dili için yapılan asenkron yazılım geliştirmeleri işlemci yükü gereken uygulamalarda performans artışı sağlamıştır. Fakat Java programlama dili için yapılan testler işlemci yükü gerektiren asenkron uygulama performansı konusunda bir fayda sağlamadığı görülmektedir. Test uygulaması geliştirilirken Java programlama dili içinde birçok farklı uygulama şekli bulunurken, biz uygulama için "Runnable" ve "FutureTask" task implementastonu üzerinden "ExecutorService" asenkron task yönetimi akışı kullanılmıştır. Java dilinde birçok asenkron programlama yaklaşımı bulunmaktadır.

Bunlar thread, CompletableFuture, ListenableFuture, EA-Async library, Oracle Fork/Join yöntemleridir. Fakat bu test uygulama özelinde bu yöntemler kullanılmadı. Tüm yöntemleri karşılaştıran bir uygulama geliştirildi. Fakat tüm yollar bu çalışma kapsamında değildir.

5.3. Öneriler

Tartışma kısmında da belirtildiği gibi java yazılım dili için diğer asenkron uygulama geliştirme yöntemleri ile de araştırma yapılabilir.

Bu yöntemlere örnekler; thread, CompletableFuture, ListenableFuture, EA-Async library, Oracle Fork/Join şeklindedir. Bu yöntemlerin hepsini içeren bir araştırma yapılabilir.

KAYNAKÇA

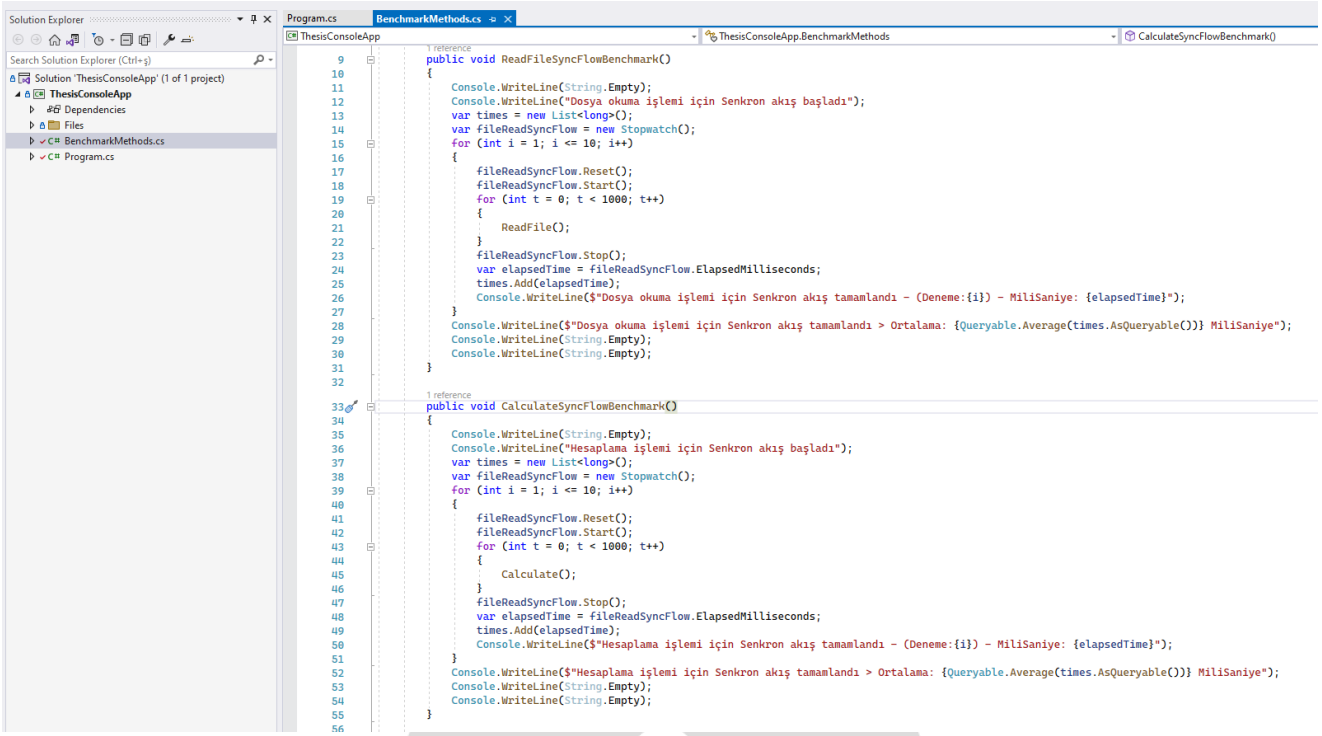
Castillo, E., Jain, N., Casas, M., Moreto, M., Schulz, M., Beivide, R., … Bhatele, A. (2019). *Optimizing computation-communication overlap in asynchronous task-based programs*. In Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP. doi:10.1145/3293883.3295720

Raman, R., Zhao, J., Sarkar, V., Vechev, M., & Yahav, E. (2010). *Efficient data race detection for async-finish parallelism*. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6418 LNCS, pp. 368–383). doi:10.1007/978-3-642-16612-9_28

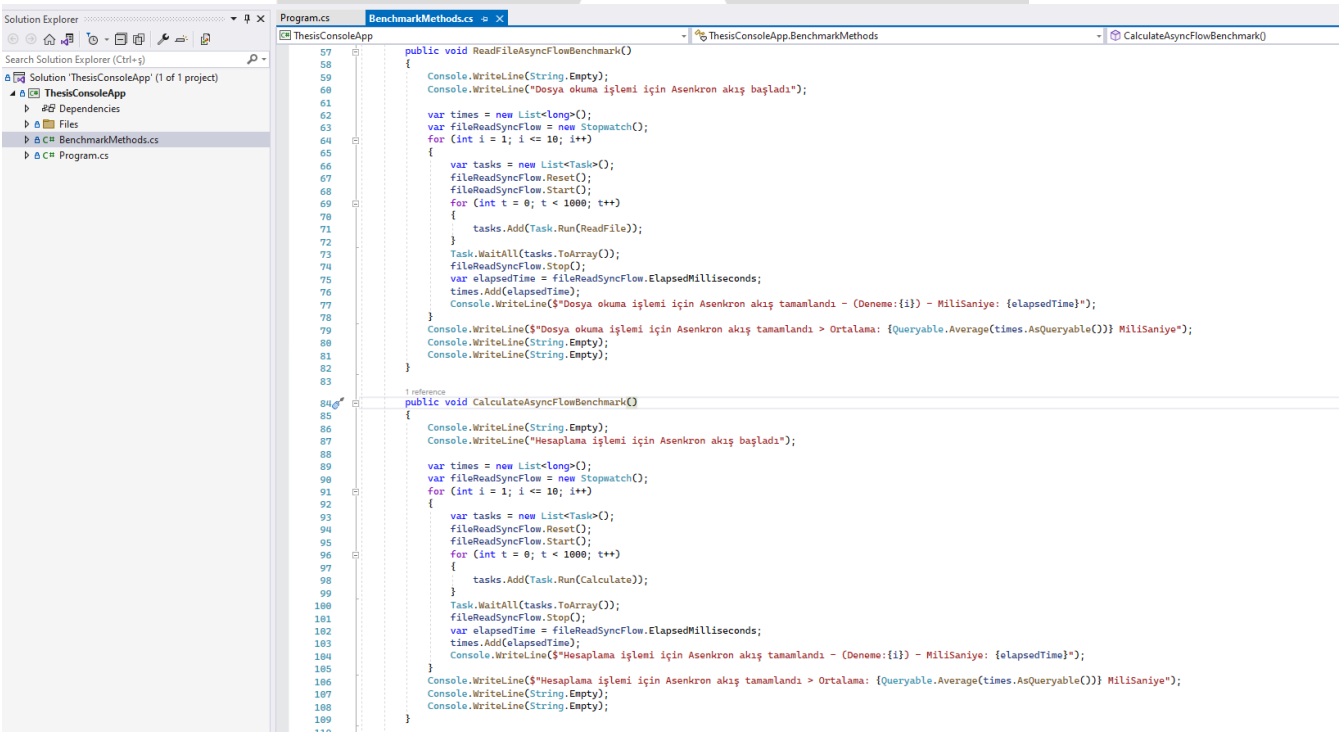
Shi, H., Zhao, Y., Zhang, B., Yoshigoe, K., & Chang, F. (2019). *Effective Parallel Computing via a Free Stale Synchronous Parallel Strategy*. *IEEE Access*, *7*. doi:10.1109/ACCESS.2019.2936820

EKLER

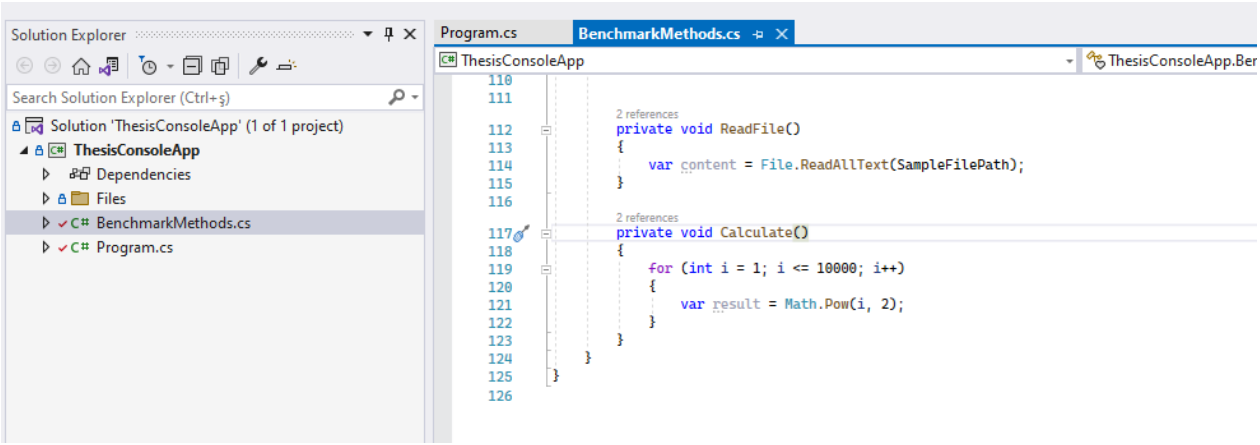
C# Uygulama Kodları



Şekil 1. C# Senkron (Sync) Akış Uygulama Kodları



Şekil 2. C# Asenkron (Async) Akış Uygulama Kodları

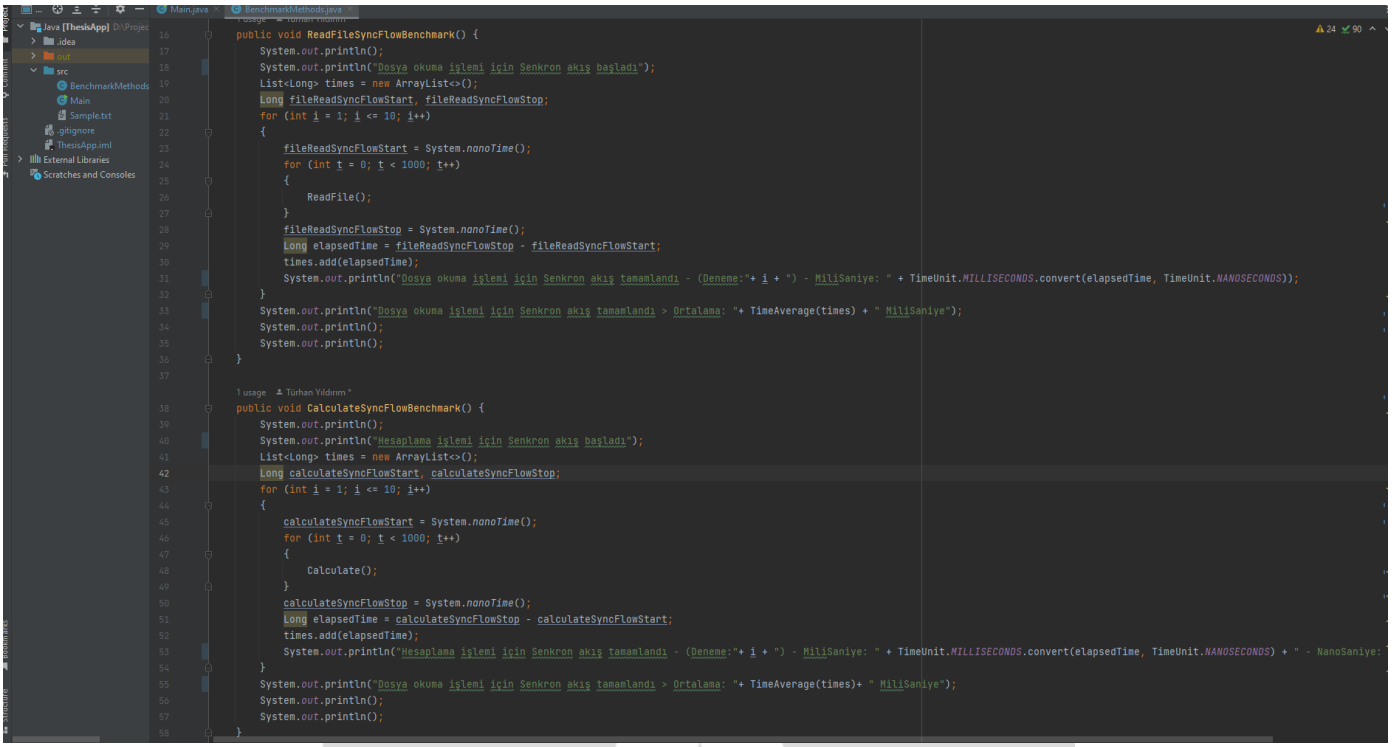


Şekil 3. C# Ana İşlem Methodlarının Kodları

```
Senkron ve Asenkron Islem Denemeleri(Dosya okuma ve hesaplama)
---------------------------------------------------------------


Dosya okuma islemi için Senkron akis basladi
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:1) - MiliSaniye: 1280
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:2) - MiliSaniye: 1058
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:3) - MiliSaniye: 1046
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:4) - MiliSaniye: 1048
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:5) - MiliSaniye: 1045
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:6) - MiliSaniye: 1046
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:7) - MiliSaniye: 1043
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:8) - MiliSaniye: 1387
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:9) - MiliSaniye: 1575
Dosya okuma islemi için Senkron akis tamamlandi - (Deneme:10) - MiliSaniye: 1566
Dosya okuma islemi için Senkron akis tamamlandi > Ortalama: 1209,4 MiliSaniye


Hesaplama islemi için Senkron akis basladi
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:1) - MiliSaniye: 239
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:2) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:3) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:4) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:5) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:6) - MiliSaniye: 238
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:7) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:8) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:9) - MiliSaniye: 237
Hesaplama islemi için Senkron akis tamamlandi - (Deneme:10) - MiliSaniye: 238
Hesaplama islemi için Senkron akis tamamlandi > Ortalama: 237,4 MiliSaniye


Hesaplama islemi için Asenkron akis basladi
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:1) - MiliSaniye: 113
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:2) - MiliSaniye: 14
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:3) - MiliSaniye: 15
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:4) - MiliSaniye: 14
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:5) - MiliSaniye: 14
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:6) - MiliSaniye: 14
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:7) - MiliSaniye: 14
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:8) - MiliSaniye: 16
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:9) - MiliSaniye: 15
Hesaplama islemi için Asenkron akis tamamlandi - (Deneme:10) - MiliSaniye: 15
Hesaplama islemi için Asenkron akis tamamlandi > Ortalama: 24,4 MiliSaniye


Dosya okuma islemi için Asenkron akis basladi
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:1) - MiliSaniye: 1280
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:2) - MiliSaniye: 1421
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:3) - MiliSaniye: 1420
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:4) - MiliSaniye: 1406
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:5) - MiliSaniye: 1398
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:6) - MiliSaniye: 1442
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:7) - MiliSaniye: 1382
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:8) - MiliSaniye: 1397
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:9) - MiliSaniye: 1384
Dosya okuma islemi için Asenkron akis tamamlandi - (Deneme:10) - MiliSaniye: 1436
Dosya okuma islemi için Asenkron akis tamamlandi > Ortalama: 1396,6 MiliSaniye
```

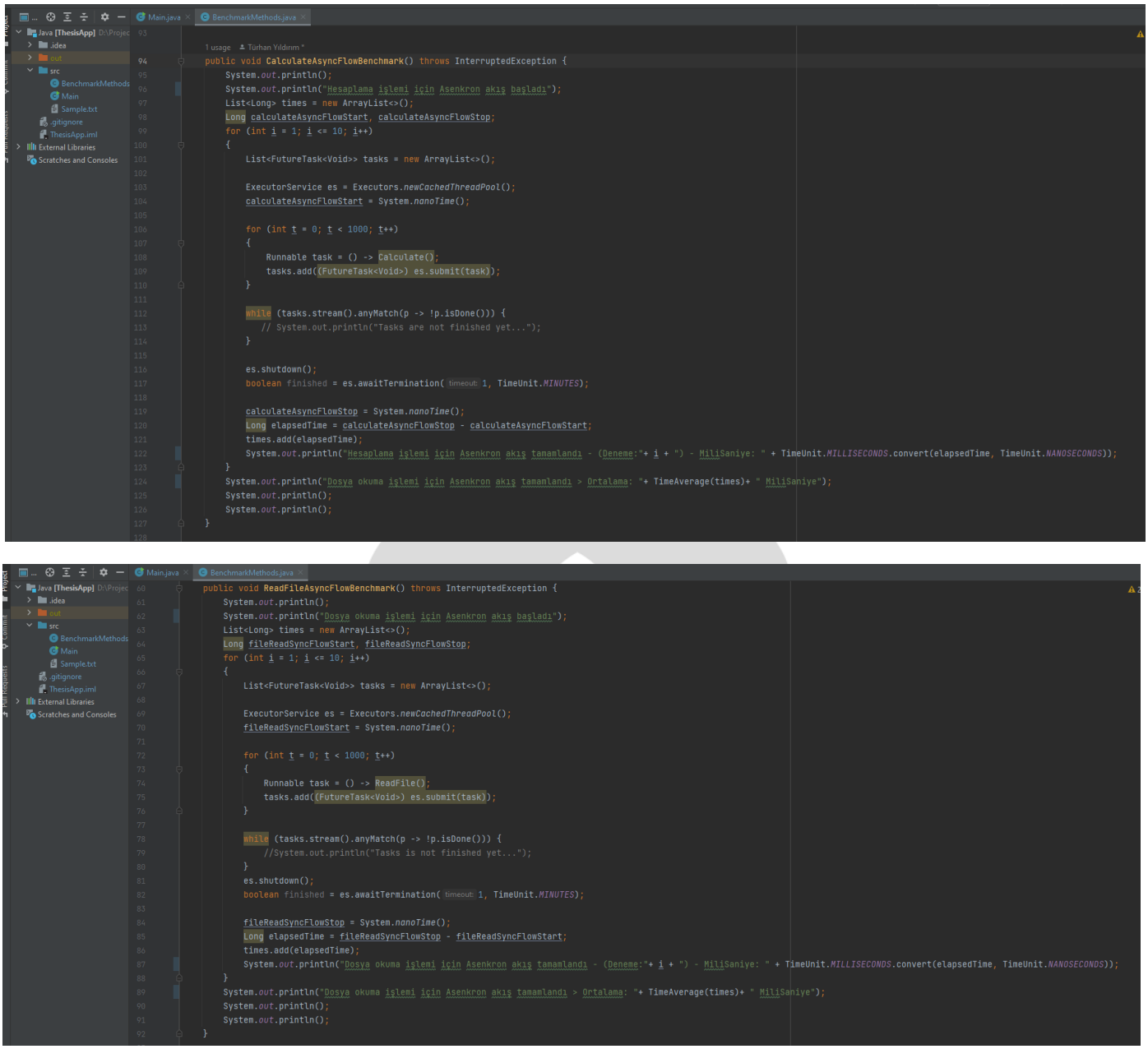
Şekil 4. C# Örnek Test Çıktısı

Java Uygulama Kodları



```
public void ReadFileSyncFlowBenchmark() {
    System.out.println();
    System.out.println("Dosya okuma işlemi için Senkron akış başladı");
    List<Long> times = new ArrayList<>();
    Long fileReadSyncFlowStart, fileReadSyncFlowStop;
    for (int i = 1; i <= 10; i++)
    {
        fileReadSyncFlowStart = System.nanoTime();
        for (int t = 0; t < 1000; t++)
        {
            ReadFile();
        }
        fileReadSyncFlowStop = System.nanoTime();
        Long elapsedTime = fileReadSyncFlowStop - fileReadSyncFlowStart;
        times.add(elapsedTime);
        System.out.println("Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:"+ i + ") - MiliSaniye: " + TimeUnit.MILLISECONDS.convert(elapsedTime, TimeUnit.NANOSECONDS));
    }
    System.out.println("Dosya okuma işlemi için Senkron akış tamamlandı > Ortalama: "+ TimeAverage(times) + " MiliSaniye");
    System.out.println();
    System.out.println();
}

1 usage  Türhan Yıldırım *
public void CalculateSyncFlowBenchmark() {
    System.out.println();
    System.out.println("Hesaplama işlemi için Senkron akış başladı");
    List<Long> times = new ArrayList<>();
    Long calculateSyncFlowStart, calculateSyncFlowStop;
    for (int i = 1; i <= 10; i++)
    {
        calculateSyncFlowStart = System.nanoTime();
        for (int t = 0; t < 1000; t++)
        {
            Calculate();
        }
        calculateSyncFlowStop = System.nanoTime();
        Long elapsedTime = calculateSyncFlowStop - calculateSyncFlowStart;
        times.add(elapsedTime);
        System.out.println("Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:"+ i + ") - MiliSaniye: " + TimeUnit.MILLISECONDS.convert(elapsedTime, TimeUnit.NANOSECONDS) + " - NanoSaniye:
    }
    System.out.println("Dosya okuma işlemi için Senkron akış tamamlandı > Ortalama: "+ TimeAverage(times)+ " MiliSaniye");
    System.out.println();
    System.out.println();
}
```

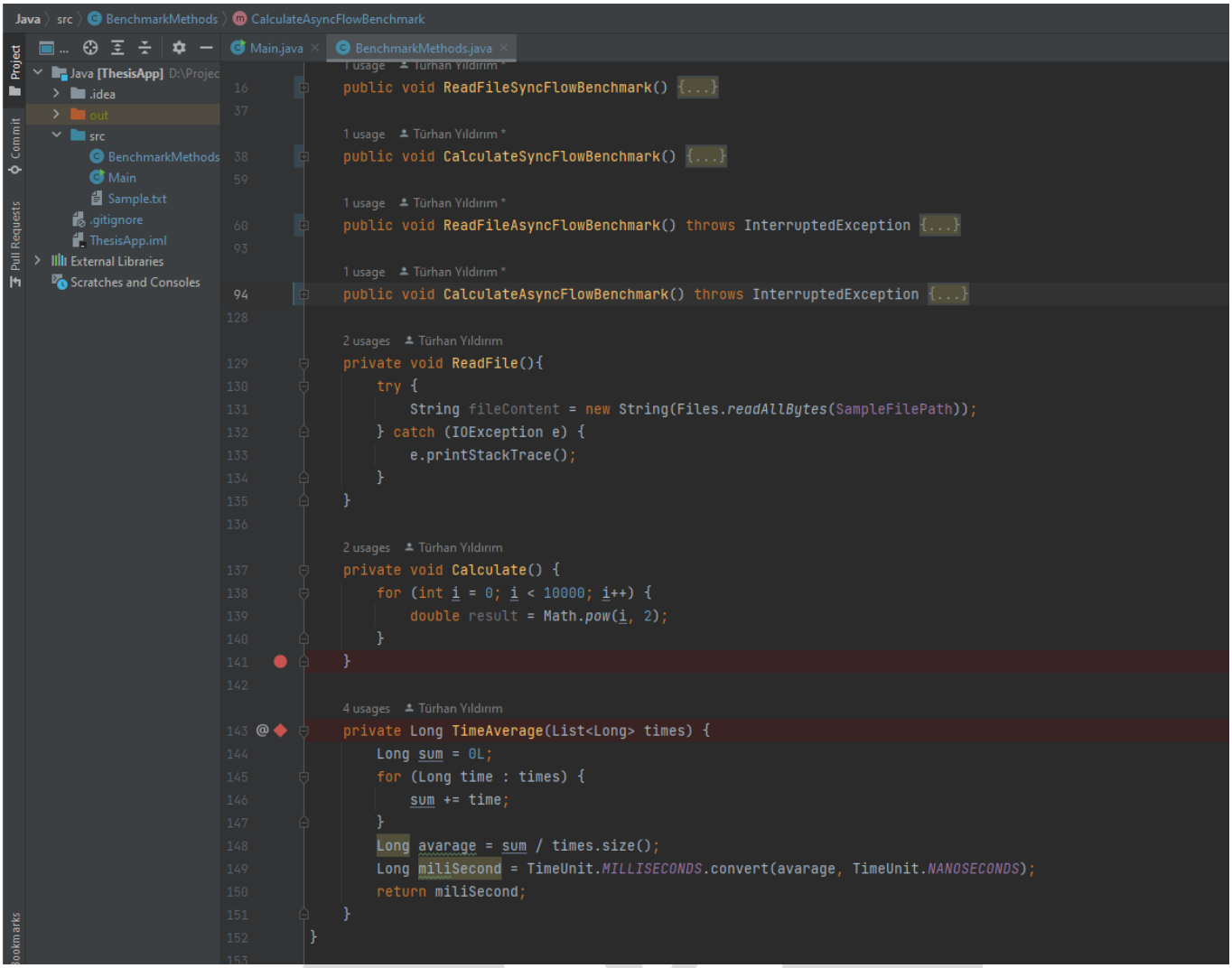
Şekil 5. Java Senkron (Sync) Akış Uygulama Kodları



```
1 usage  Türhan Yıldırım *
public void CalculateAsyncFlowBenchmark() throws InterruptedException {
    System.out.println();
    System.out.println("Hesaplama işlemi için Asenkron akış başladı");
    List<Long> times = new ArrayList<>();
    Long calculateAsyncFlowStart, calculateAsyncFlowStop;
    for (int i = 1; i <= 10; i++)
    {
        List<FutureTask<Void>> tasks = new ArrayList<>();

        ExecutorService es = Executors.newCachedThreadPool();
        calculateAsyncFlowStart = System.nanoTime();

        for (int t = 0; t < 1000; t++)
        {
            Runnable task = () -> Calculate();
            tasks.add((FutureTask<Void>) es.submit(task));
        }

        while (tasks.stream().anyMatch(p -> !p.isDone())) {
            // System.out.println("Tasks are not finished yet...");
        }

        es.shutdown();
        boolean finished = es.awaitTermination( timeout: 1, TimeUnit.MINUTES);

        calculateAsyncFlowStop = System.nanoTime();
        Long elapsedTime = calculateAsyncFlowStop - calculateAsyncFlowStart;
        times.add(elapsedTime);
        System.out.println("Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:"+ i + ") - MiliSaniye: " + TimeUnit.MILLISECONDS.convert(elapsedTime, TimeUnit.NANOSECONDS));
    }
    System.out.println("Dosya okuma işlemi için Asenkron akış tamamlandı > Ortalama: "+ TimeAverage(times)+ " MiliSaniye");
    System.out.println();
    System.out.println();
}
```

```
public void ReadFileAsyncFlowBenchmark() throws InterruptedException {
    System.out.println();
    System.out.println("Dosya okuma işlemi için Asenkron akış başladı");
    List<Long> times = new ArrayList<>();
    Long fileReadSyncFlowStart, fileReadSyncFlowStop;
    for (int i = 1; i <= 10; i++)
    {
        List<FutureTask<Void>> tasks = new ArrayList<>();

        ExecutorService es = Executors.newCachedThreadPool();
        fileReadSyncFlowStart = System.nanoTime();

        for (int t = 0; t < 1000; t++)
        {
            Runnable task = () -> ReadFile();
            tasks.add((FutureTask<Void>) es.submit(task));
        }

        while (tasks.stream().anyMatch(p -> !p.isDone())) {
            //System.out.println("Tasks is not finished yet...");
        }
        es.shutdown();
        boolean finished = es.awaitTermination( timeout: 1, TimeUnit.MINUTES);

        fileReadSyncFlowStop = System.nanoTime();
        Long elapsedTime = fileReadSyncFlowStop - fileReadSyncFlowStart;
        times.add(elapsedTime);
        System.out.println("Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:"+ i + ") - MiliSaniye: " + TimeUnit.MILLISECONDS.convert(elapsedTime, TimeUnit.NANOSECONDS));
    }
    System.out.println("Dosya okuma işlemi için Asenkron akış tamamlandı > Ortalama: "+ TimeAverage(times)+ " MiliSaniye");
    System.out.println();
    System.out.println();
}
```

Şekil 6. Java Asenkron (Async) Akış Uygulama Kodları



Şekil 7. Java Ana İşlem Methodlarının Kodları

```
C:\Users\turha\.jdks\corretto-1.8.0_332\bin\java.exe ...
Senkron ve Asenkron İşlem Denemeleri(Dosya okuma ve hesaplama)
-------------------------------------------------------------


Dosya okuma işlemi için Senkron akış başladı
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:1) - MiliSaniye: 1080
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:2) - MiliSaniye: 828
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:3) - MiliSaniye: 651
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:4) - MiliSaniye: 646
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:5) - MiliSaniye: 645
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:6) - MiliSaniye: 644
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:7) - MiliSaniye: 642
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:8) - MiliSaniye: 648
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:9) - MiliSaniye: 643
Dosya okuma işlemi için Senkron akış tamamlandı - (Deneme:10) - MiliSaniye: 644
Dosya okuma işlemi için Senkron akış tamamlandı > Ortalama: 707 MiliSaniye



Hesaplama işlemi için Senkron akış başladı
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:1) - MiliSaniye: 1 - NanoSaniye: 1586100
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:2) - MiliSaniye: 0 - NanoSaniye: 22200
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:3) - MiliSaniye: 0 - NanoSaniye: 20900
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:4) - MiliSaniye: 0 - NanoSaniye: 21800
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:5) - MiliSaniye: 0 - NanoSaniye: 22000
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:6) - MiliSaniye: 0 - NanoSaniye: 22300
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:7) - MiliSaniye: 0 - NanoSaniye: 20100
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:8) - MiliSaniye: 0 - NanoSaniye: 22300
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:9) - MiliSaniye: 0 - NanoSaniye: 22000
Hesaplama işlemi için Senkron akış tamamlandı - (Deneme:10) - MiliSaniye: 0 - NanoSaniye: 22100
Dosya okuma işlemi için Senkron akış tamamlandı > Ortalama: 0 MiliSaniye



Hesaplama işlemi için Asenkron akış başladı
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:1) - MiliSaniye: 36
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:2) - MiliSaniye: 3
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:3) - MiliSaniye: 2
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:4) - MiliSaniye: 2
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:5) - MiliSaniye: 2
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:6) - MiliSaniye: 3
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:7) - MiliSaniye: 2
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:8) - MiliSaniye: 1
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:9) - MiliSaniye: 1
Hesaplama işlemi için Asenkron akış tamamlandı - (Deneme:10) - MiliSaniye: 1
Dosya okuma işlemi için Asenkron akış tamamlandı > Ortalama: 5 MiliSaniye



Dosya okuma işlemi için Asenkron akış başladı
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:1) - MiliSaniye: 811
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:2) - MiliSaniye: 804
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:3) - MiliSaniye: 864
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:4) - MiliSaniye: 693
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:5) - MiliSaniye: 834
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:6) - MiliSaniye: 631
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:7) - MiliSaniye: 703
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:8) - MiliSaniye: 636
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:9) - MiliSaniye: 664
Dosya okuma işlemi için Asenkron akış tamamlandı - (Deneme:10) - MiliSaniye: 643
Dosya okuma işlemi için Asenkron akış tamamlandı > Ortalama: 728 MiliSaniye
```

Şekil 8. Java Örnek Test Çıktısı