

CSE 222 HOMEWORK #4

HASAN MEN

131044009

TABLE OF CONTENTS

1. Gereksinimler.....	2
1.1. Ön inceleme.....	2
1.2. Gereksinim ayrıntıları.....	2
2. Analiz ve Soruna Yaklaşım.....	3
3. Sınıf Diyagramları.....	4
4. Use Case.....	7
5. Testler.....	8

1. Gereksinimler

1.1. On İnceleme

Programımız bir assembly converter gibi işleyecek. Infix notasyonları alıp postfixe ordanda assemblyye çevirip kullanılabilir bir çıktı vermesi hedeflenmektedir.

1.2. Gereksinim ayrıntıları

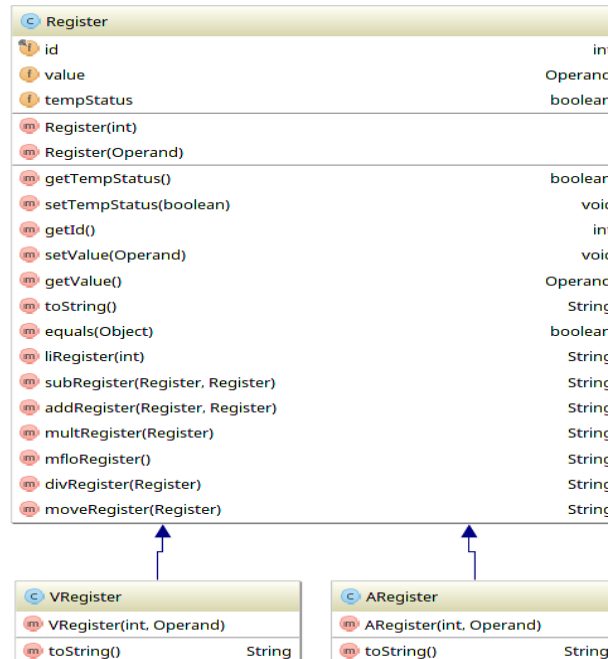
- 1.2.1. Dosya işlemlerini kolaylaştırmak için kendi okuma ve yazma classlarımı yazdım.
- 1.2.2. Infix olarak alınan kodların satır satır postfixe çevirilip daha sonradan assembly e çevirilmesi lazım.
- 1.2.3. Assembler classımız olmalı ve postfixler üzerinde işlemlerini yürütebilmeli.
- 1.2.4. Assemblerde kullanılacak registerler belirli sayıda Stack yapısı ile assemblerde depolanmalı.
- 1.2.5. Assemblerde operand stacki olmalı ve her token operand operator gorulup işlem yapılana kadar depolanmalı.
- 1.2.6. Kullanılabilir registerlere ek olarak birde kullanılan registerleri göz altında tutmak için ArrayList list ile used registerler depolandı.
- 1.2.7. Assemblerde kullanılmak üzere t türü registerlerden v ve a türünden registerler türetilmeli.
- 1.2.8. Olağan tüm hatalar tutulmalı

2. Analiz ve Soruna Yaklaşım

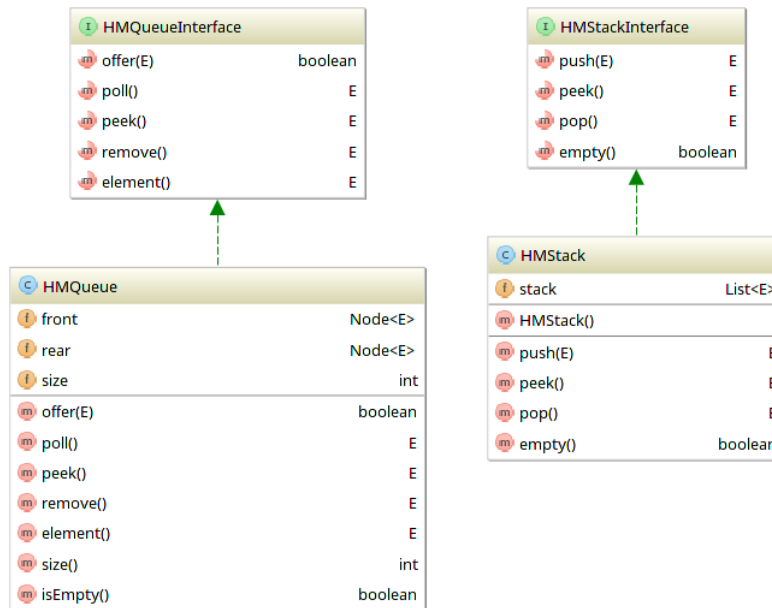
Dosyadan aldığım operandları satır satır assembler içinde depolayacağım. Daha sonra her satır üzerinde ayrı ayrı işlemler yaparak aynı zamanda registerlerle senkronize olmalıyım. Kullanılan her register kullanım listesine eklenip, uygun registerlerden çıkarılmalı. Temp olarak belirtilen registerler işleri biter bitmez yok edilip resource kullanımı en aza indirilmeli. Postfix quesinden gelen string tokenlerine ayrılıp işleme alındıktan sonra operand atamalarında geçerlilik dikkate alınmalı.

3. Sınıf Diyagramları

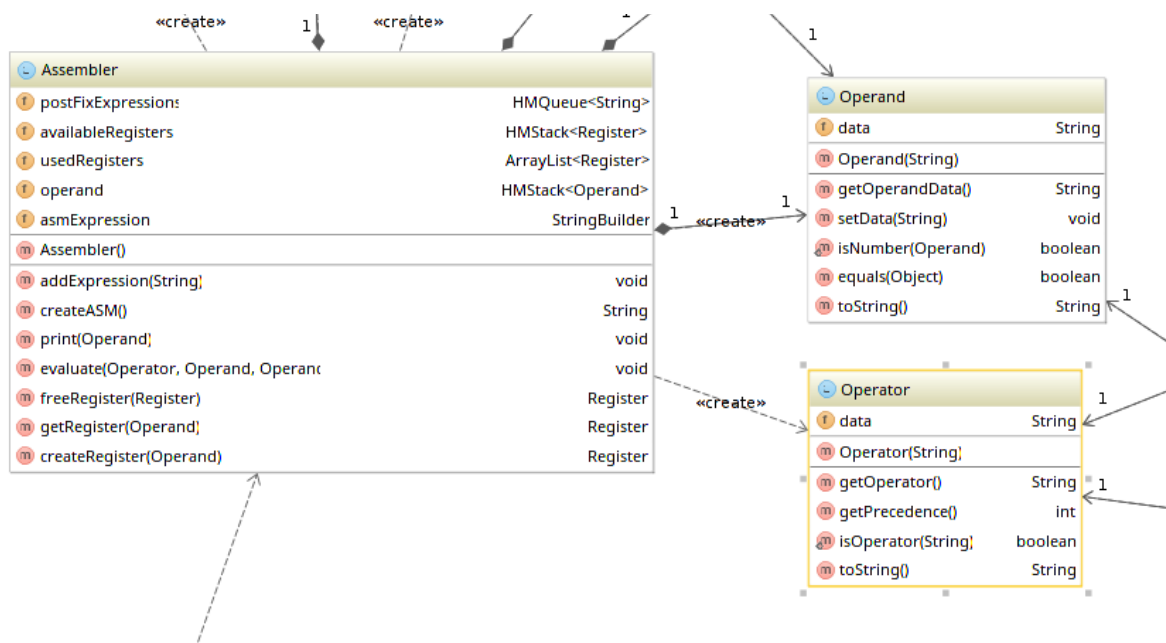
3.1 Register Sınıfı



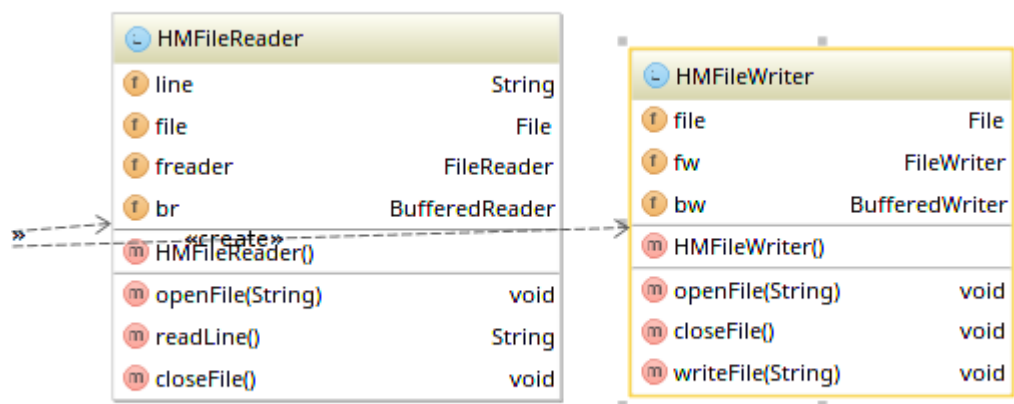
3.2 Queue ve Stack Sınıfları



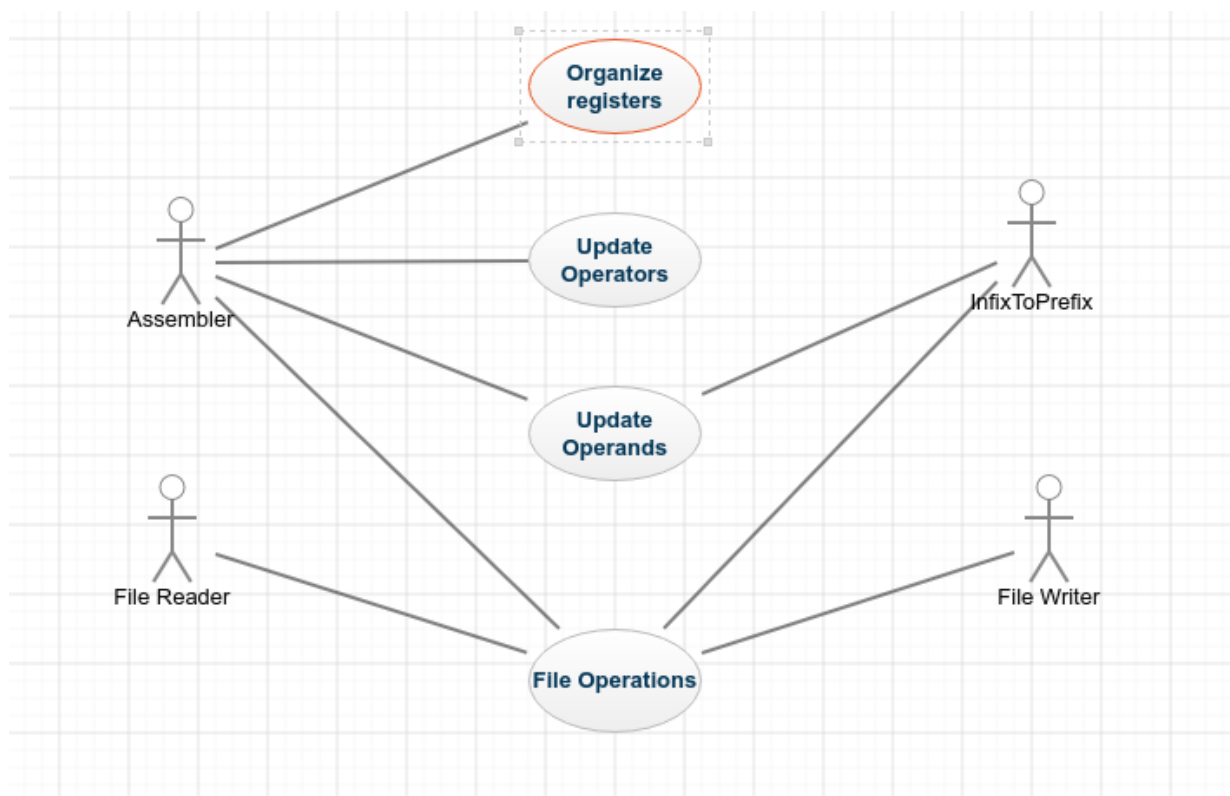
3.2 Assembler, Operands and Operators



3.3 File Reader and Writer Sınıfları

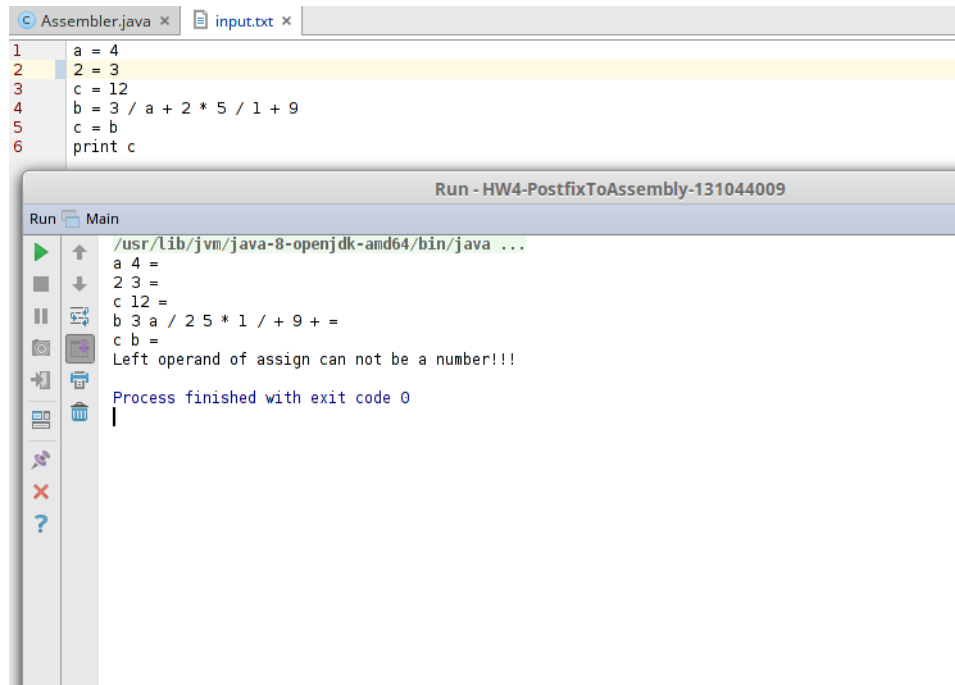


4. Use Case



5. Testler

Test 1 (Integer bir sayiya atama)



The screenshot shows an IDE with two tabs: `Assembler.java` and `input.txt`. The `Assembler.java` file contains the following code:

```

1  a = 4
2  2 = 3
3  c = 12
4  b = 3 / a + 2 * 5 / 1 + 9
5  c = b
6  print c

```

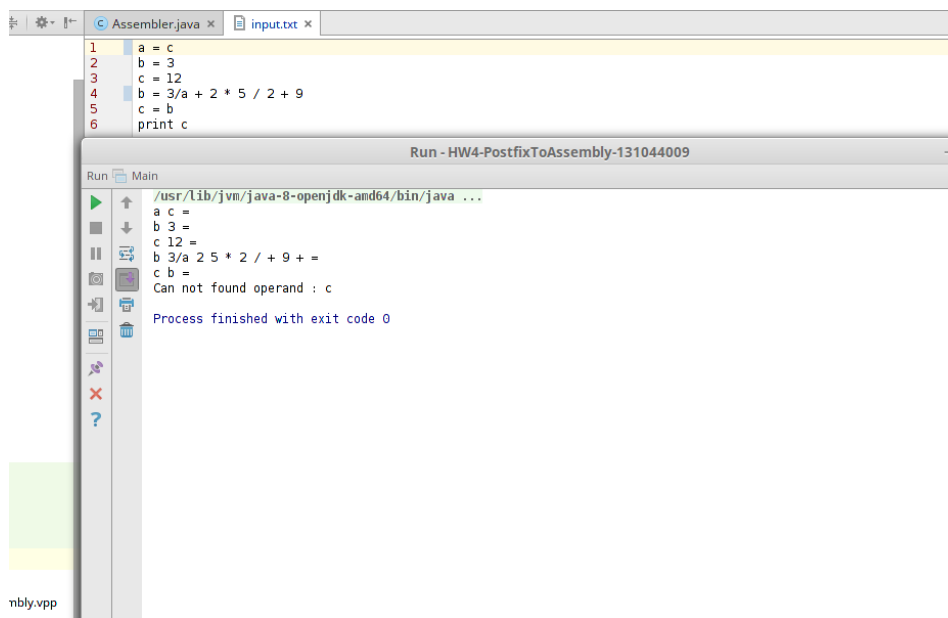
Below the code editor, a run window titled "Run - HW4-PostfixToAssembly-131044009" displays the execution output:

```

Run Main
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
a 4 =
2 3 =
c 12 =
b 3 a / 2 5 * 1 / + 9 + =
c b =
Left operand of assign can not be a number!!!
Process finished with exit code 0

```

Test 2 (Hatalı karakter ataması)



The screenshot shows an IDE with two tabs: `Assembler.java` and `input.txt`. The `Assembler.java` file contains the following code:

```

1  a = c
2  b = 3
3  c = 12
4  b = 3/a + 2 * 5 / 2 + 9
5  c = b
6  print c

```

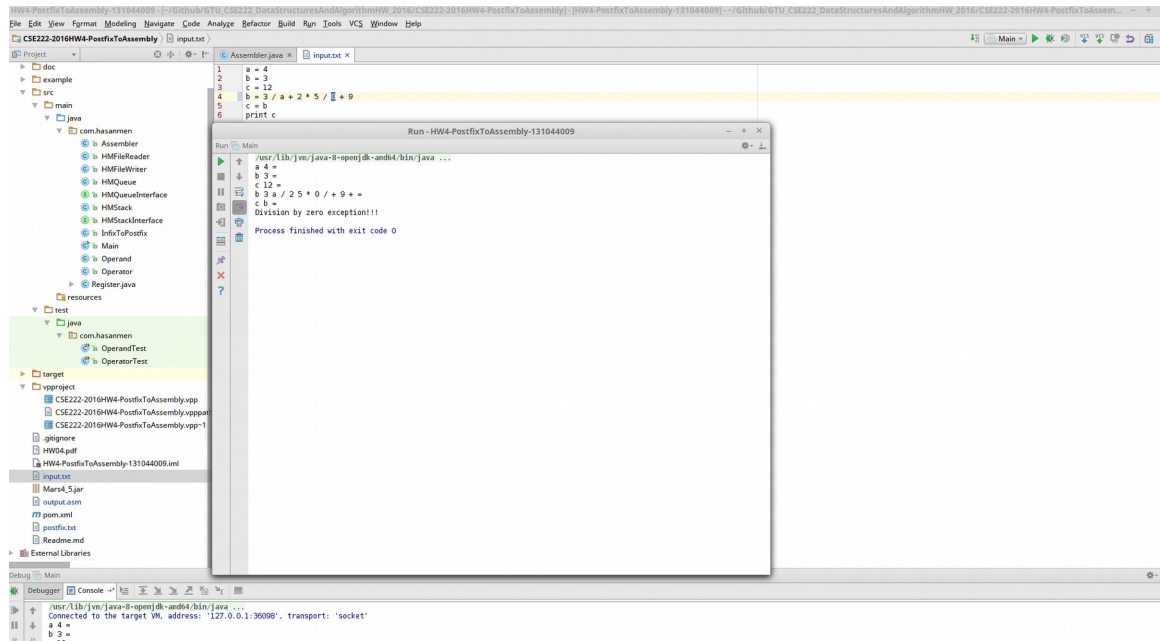
Below the code editor, a run window titled "Run - HW4-PostfixToAssembly-131044009" displays the execution output:

```

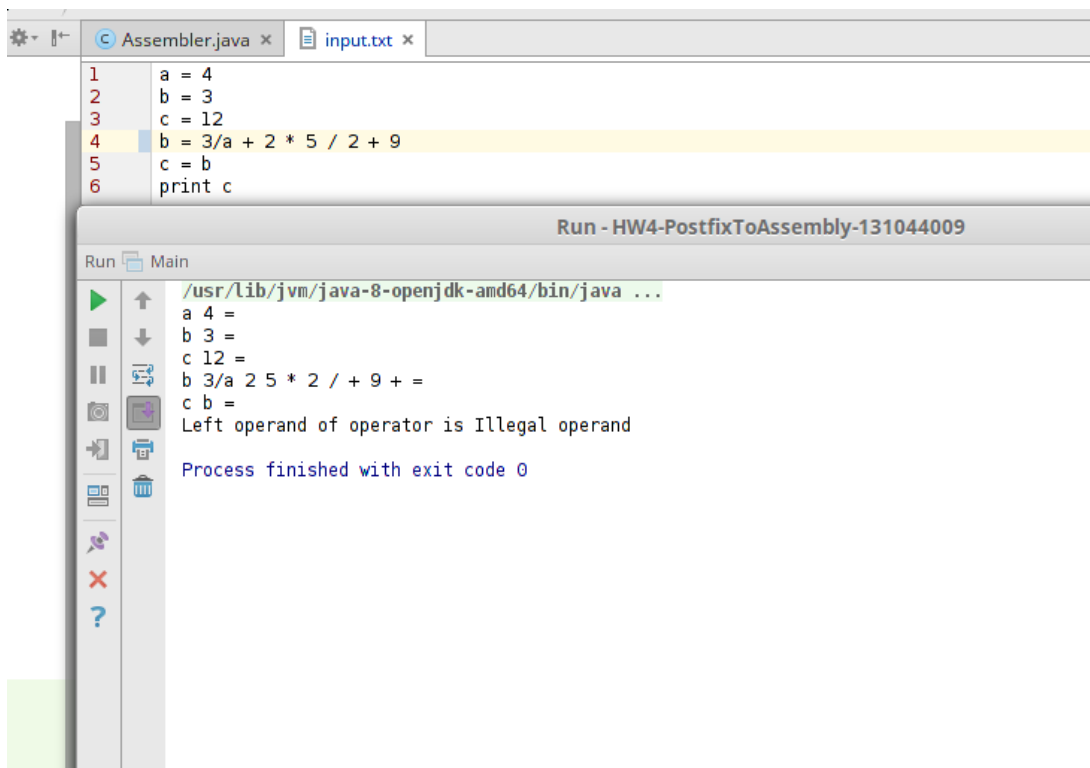
Run Main
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
a c =
b 3 =
c 12 =
b 3/a 2 5 * 2 / + 9 + =
c b =
Can not found operand : c
Process finished with exit code 0

```

Test 3 (Division by zero)



Test 4 (Hatalı operand kullanımı)



Çalışan bir örnek

The screenshot displays the Mars MIPS simulator interface. The main window shows the assembly code generated from the Java source code. The assembly code is as follows:

```

1  a = 2
2  b = 3
3  c = 12
4  b = 3 / a + 2 * 5 / 2 + 9
5  c = 9
6  b = b * c
7  print b

```

The assembly code is shown in the main window and a separate window. The assembly code is as follows:

```

1  li $t0, 2
2  li $t1, 3
3  li $t2, 12
4  li $t3, 3
5  div $t3, $t0
6  mflo $t3
7  li $t4, 2
8  li $t5, 5
9  mult $t4, $t5
10 mflo $t4
11 li $t6, 2
12 li $t7, 5
13 mult $t6, $t7
14 mflo $t6
15 li $t8, 2
16 div $t3, $t8
17 mflo $t3
18 add $t3, $t3, $t6
19 li $t9, 4
20 add $t3, $t3, $t9
21 move $t1, $t3
22
23 li $t2, 9
24
25 mult $t1, $t2
26 mflo $t3
27 move $t1, $t3
28
29 move $a0, $t1
30 li $v0, 1
31 syscall

```

The memory segment is shown in the bottom right window. The memory segment is as follows:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	0	0	0	0	0	0	0	0
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0

The bottom panel shows the program's execution progress. The program is finished running (dropped off bottom).