

# Algorithms

## CSE464/564

### Familial Matching in a Fraternal Environment – Set up

Meisam Amjad, Prasidh Arora, Megan Moore

---

#### Public class Matching<T>

The matching class written in JAVA reads data from a file containing a list of Children and Parents, and matches them up based on their preference. This class is designed for Automated Pairings in a Fraternal Familial Environment project, but it can also be used for similar pairing problems such as in an adoption agency.

This class uses an adaptation of the [Gale Shapley algorithm](#) which matches the same sized Children and Parent lists with the same sized preference lists. However, this class builds on the algorithm by creating a more efficient class that can be used when Children and Parents having different preference lists of different sizes. In addition, this class can pair up Parents and Children when a Parent wants to be matched with more than one Child (See [Parent](#) for setting more than one child and [Child](#) Class).

This class is optimized to use Integer type as a generic type <T extends as Integer>, allowing for better efficiency. However, it only works with Children's ID number and Parent's Roll number rather than their name (type String). Therefore, declaring the class would be as follows:

*Matching<Integer> M;*

There are a variety of constructors for this class. Below, you can see an example of 2 constructors for defining a Matching object ([See list of all Constructor](#)).

*Matching<Integer> M = new Matching<Integer>();*

Using a default constructor for having all fields to null.

*Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("data.txt"));*

The matching class uses the [FileManager Class](#) for reading all data from a file (as an [input data](#)). It then reads data from that class for initializing all the [Fields](#). The object M now holds all data from the given file.

Instead of constructors in this class, users can have access to the Fields which hold the lists of Parents and Children with details that are read from a given file. For getting the desired result, the pairing method ([See list of all Methods](#)) needs to be run (*fraternalPairing()* method). This will allow the user to have access to the matched data directly or even displayed on the screen.

Now, before any further explanation let's look at all details in this class:

#### Field Summary:

Fields	
<code>List&lt;Integer&gt;</code> <code>getParents()</code>	List of all Parents by their Roll Number.
<code>List&lt;Integer&gt;</code> <code>getChildren()</code>	List of all Children by their ID Number.
<code>HashMap&lt;Integer, Parent&lt;T&gt;&gt;</code> <code>getParentMap()</code>	List of all Parents<Key, Value>. The key is the Parent's Roll Number and the Value is the Parent class containing all details regarding that Parent.
<code>HashMap&lt;Integer, Child&lt;T&gt;&gt;</code> <code>getChildrenMap()</code>	List of all Children<Key, Value>. The key is the Child's ID Number and the Value is the Child class containing all details regarding that Child.
<code>HashMap&lt;Integer, Integer&gt;</code> <code>getPairs()</code>	List of all matched<Key, Value>. This list is produced after the <code>fraternalPairing()</code> method gets executed once. The Key is the Child's ID and the Value is Parent's Roll Number.

#### Constructor Summary:

Constructors:	
<code>Matching&lt;Integer&gt; ()</code>	Initializes a Matching object so that it represents an empty Matching object.
<code>Matching&lt;Integer&gt;(FileManager&lt;Integer&gt; f)</code>	Initializes a Matching object with all data read by the given FileManager class.
<code>Matching&lt;Integer&gt;(Matching&lt;Integer&gt; M)</code>	Initializes a Matching object by making a deep copy of the given M.

## Method Summary:

Methods:	
Modifier and Type	Method and Description
HashMap<Integer, Integer>	<a href="#">fraternalPairing()</a> Returns a List of all pairs that have been matched.
void	<a href="#">printPairs()</a> Displays all matches on the display. The fraternalPairing() method should have been run before this method, otherwise there is no result to be displayed.

## Setting up the Environment

This class is designed to be used easily, even for beginners. Using this class contains three easy steps:

1. Defining a Matching object.
2. Running the fraternal pairing method.
3. Seeing the result.

We will see many examples of using this class later in this document, but the most important part of using this class is preparing the data (inputs) and passing them into the class. As mentioned, all inputs pass in as a file and in the next section we will explain the details.

### ● Input Data:

Since the number of inputs provided in the file could be quite large, this class is only designed to read data from a file by using the [FileManager](#) Class. However, the data needs to be written in a specific format to be read by the FileManager. In this section we show different sample data that can be used for this class.

All data needs to be written in a .txt file or .csv file as following:

### For a Child:

Capital character 'C'(as a command), child's ID number, child's Name, list of child's preferred parents in order separated by comma. Note that we only use Parent's Roll number for child's preferences. For example:

*C, 1, Alex, 5, 7, 6, 2, 3, 4, 1*

1. The line starts with capital 'C' as a command for FileManager class to recognize that this data belongs to a Child.
2. Child's ID Number.
3. Child's Name.
4. Child's first preferred Parent.
5. Child's second preferred Parent.
6. The rest of Child's preferred Parents.

Again, note that all elements are separated by commas.

#### For a Parent:

Capital character 'P'(as a command), Parent's Roll Number, Parent's Name, Parent's available children ([see here for details](#)), list of parent's preferred children separated by comma.

For example:

*P, 1, W1, 1, 5, 3, 4, 6, 7, 1, 2*

1. The line starts with capital 'P' as a command for FileManager class to shows that this data belongs to a Parent.
2. Parent's Roll Number.
3. Parent's Name.
4. Parent's Available Children. ([see here for details](#))
5. Parent's first preferred Child.
6. Parent's second preferred Child.
7. The rest of Parent's preferred Children.

All elements must be separated by commas.

Below we can see all data from the file 'data04.txt' as an example:

*C, 1, M1, 5, 7, 6, 2, 3, 4, 1*

*C, 2, M2, 6, 2, 5, 1, 3, 7, 4*

*C, 3, M3, 4, 3, 7, 2, 1, 5, 6*

*C, 4, M4, 7, 1, 2, 6, 3, 4, 5*

*C, 5, M5, 5, 2, 3, 4, 6, 1, 7*

*C, 6, M6, 3, 4, 5, 6, 7, 1, 2*

*C, 7, M7, 2, 1, 4, 7, 2, 3, 6*

*P, 1, W1, 1, 5, 3, 4, 6, 7, 1, 2*

*P, 2, W2, 1, 1, 2, 3, 5, 7, 4, 6*

*P, 3, W3, 1, 6, 7, 5, 3, 2, 1, 4*

*P, 4, W4, 1, 6, 5, 7, 2, 1, 4, 3*

*P, 5, W5, 1, 2, 1, 6, 4, 3, 5, 7*

*P, 6, W6, 1, 7, 4, 5, 3, 2, 1, 6*

*P, 7, W7, 1, 3, 4, 7, 5, 6, 2, 1*

As you can see all elements are separated by commas and we only use Roll numbers and ID numbers for displaying list of Parent's or Child's preferences.

Another example, we can look at the data inside the 'data02.txt' file:

*C, 1, SWO, 16, 14,  
C, 2, DIW, 17,  
C, 3, HYI, 12, 16, 14, 13  
C, 4, JFB, 11, 17, 15, 12  
C, 5, NGU, 12, 15, 17  
C, 6, GBK, 12, 13, 14, 11  
C, 7, RKT, 12, 14, 11, 13  
C, 8, VJE, 16, 12, 11, 15  
C, 9, BHE, 13, 14, 15  
C, 10, ALP, 11, 15, 17, 12*

*P, 11, NVE, 2, 5, 8, 9, 2  
P, 12, HVY, 2, 6, 7, 1, 10  
P, 13, XLH, 1, 4, 10, 1  
P, 14, LKV, 2, 7, 9, 6  
P, 15, XCE, 1, 10, 7, 3, 6  
P, 16, SBG, 1, 5, 3, 8, 1  
P, 17, FWX, 1, 6, 7, 4, 9*

The sample above, demonstrates 2 big advantages of using this class for finding matches:

1. As you can see some Parents want more than one Child which in this example, the maximum is 2. For instance, in the following line:

*P, 11, NVE, 2, 5, 8, 9, 2*

The Parent called NVE wants to be matched with two Children (availableChild = 2).

2. The number of preferences for each Child and Parent are different. For instance, in the following line:

*C, 2, DIW, 17*

The child only has one preferred Parent. Another example would be the following line:

*C, 1, SWO, 16, 14*

The child has only 2 preferred parents, while other lines have 4 or 5 preferences.

- **Defining a Matching Object**

This Class has three [constructors](#). Excluding the default constructor that initializes an empty object, we only need to use the line below for defining a Matching object along with initializing all Fields with real data from the given file:

```
Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("data02.txt"));
```

- **Exploring the Data**

Using 'data02.txt' as an example for this context, we want to define a matching object and look at all data using only the matching object:

*Data02.txt:*

```
C, 1, SWO, 16, 14,  
C, 2, DIW, 17,  
C, 3, HYI, 12, 16, 14, 13  
C, 4, JFB, 11, 17, 15, 12  
C, 5, NGU, 12, 15, 17  
C, 6, GBK, 12, 13, 14, 11  
C, 7, RKT, 12, 14, 11, 13  
C, 8, VJE, 16, 12, 11, 15  
C, 9, BHE, 13, 14, 15  
C, 10, ALP, 11, 15, 17, 12  
  
P, 11, NVE, 2, 5, 8, 9, 2  
P, 12, HVY, 2, 6, 7, 1, 10  
P, 13, XLH, 1, 4, 10, 1  
P, 14, LKV, 2, 7, 9, 6  
P, 15, XCE, 1, 10, 7, 3, 6  
P, 16, SBG, 1, 5, 3, 8, 1  
P, 17, FWX, 1, 6, 7, 4, 9
```

We define a Matching Object and call it M:

```
Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("data02.txt"));
```

Now, let's look at the list of all Children. For that, we use the *getChildren()* method and a *for each* loop for looping through all ID numbers one by one and displaying them:

```
1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data02.txt"));
8         for (int ID: M.getChildren())
9             System.out.print(ID + "\t");
10    }
11
12 }
13
```

Problems Javadoc Declaration Console X Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2018,

1 2 3 4 5 6 7 8 9 10

**Figure 1 .** Using getChildren() method.

You can see at the bottom of figure 1, the result gets printed as:

1      2      3      4      5      6      7      8      9      10

Which are ID Numbers for 10 children inside the 'data02.txt' file.

Similarly, we can use the *getParents()* method to return a list of the Roll Numbers:

```
1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data02.txt"));
8         for (int RollNumber: M.getParents())
9             System.out.print(RollNumber + "\t");
10    }
11
12 }
13
```

Problems Javadoc Declaration Console X Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2018,

11 12 13 14 15 16 17

**Figure 2 .** Using getParents() method.

As seen in figure 2, the `getParents()` method uses a *for each* loop to display all the Roll Numbers as follows:

11      12      13      14      15      16      17

Furthermore, we can see the Parents and Children with all their details as they are represented in 'data02.txt' file. The methods `getParentMap()` and `getChildrenMap()` gives us the ability to look through all details for each Parent or Child. For demonstration, Figure 3 shows the code for displaying the information of each parent in a given file:

```

1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data02.txt"));
8         for (int RollNumber: M.getParents())
9             System.out.print(M.getParentsMap().get(RollNumber));
10    }
11 }
12
13

```

```

<terminated> Group_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_77.jdk/Contents/Home/bin/java (Apr 29, 2018)
Roll Number : 11
Name : NVE
Available Child : 2
Ranked Children :
5 8 9 2
Roll Number : 12
Name : HVY
Available Child : 2
Ranked Children :
6 7 1 10
Roll Number : 13
Name : XLH
Available Child : 1
Ranked Children :
4 10 1
Roll Number : 14
Name : LKV
Available Child : 2
Ranked Children :
7 9 6
Roll Number : 15
Name : XCE
Available Child : 1
Ranked Children :
10 7 3 6
Roll Number : 16
Name : SBG
Available Child : 1
Ranked Children :
5 3 8 1
Roll Number : 17
Name : FWX
Available Child : 1
Ranked Children :
6 7 4 9

```

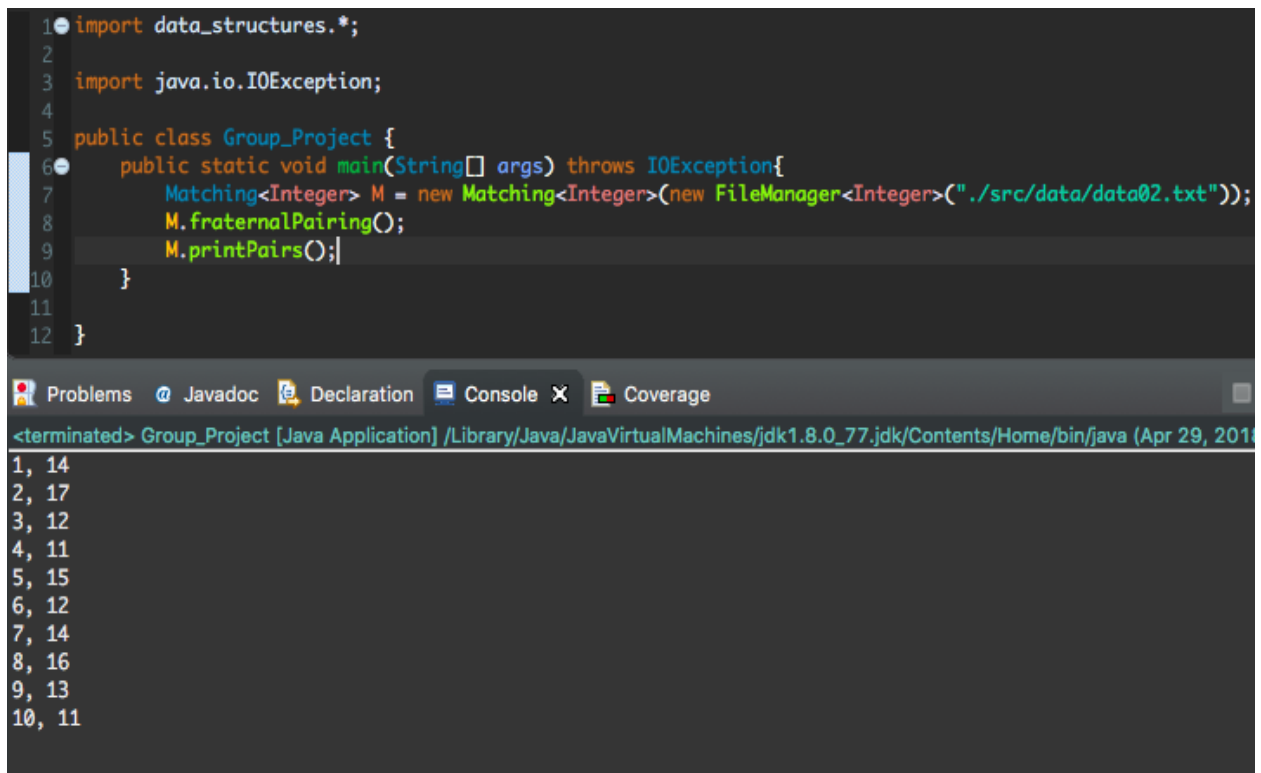
**Figure 3 .** Using `getParent()` and `getParentMap()` for showing all Parents with details.

## ● Producing Results

Thus far, we have gotten familiar with the data we need to give and how we can explore each piece of data using Matching class. Thus, we must produce the result which is a list of matched



Pairs. After defining a *Matching* object, we can now produce the results. The *fraternalPairing()* method goes through all of the data and matches them up based on their desired preferences. Since, this algorithm uses the modified Gale Shapley algorithm, the theoretical time would be  $O(n^2)$ . However, since this class uses a different data structure, it can produce the results in a faster time (Read the report for Automated Pairings in a Fraternal Familial Environment project). Figure 4 demonstrates the code for producing the results along with a displayed example.



```

1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data02.txt"));
8         M.fraternalPairing();
9         M.printPairs();
10    }
11
12 }

```

Problems Javadoc Declaration Console Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2014)

```

1, 14
2, 17
3, 12
4, 11
5, 15
6, 12
7, 14
8, 16
9, 13
10, 11

```

**Figure 4 .** Using *fraternalPairing()* and *printPairs()* for producing and displaying the results.

In figure 4, we can see the pairs in *<key, value>* order where the *key* is the ID number of the Child and the *value* is the Roll Number of the Parent.

We can see that the parent with Roll Number 11 got paired with two Children (Child 4 and 10) as we discussed and showed previously in the [Input Data](#) section.

The method *printPairs()* displays the result on the screen, but for our purposes, it would be more effective to have access to the produced data. The method *getPairs()* returns a *HashMap<Integer, Integer>* holding all produced pairs where the key is the child's ID numbers. In figure 5, we also show another way of displaying the result by using *getPairs()* method and a loop.

```

1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException {
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data02.txt"));
8         M.fraternalPairing();
9         for (int ID:M.getChildren())
10             System.out.println(ID + ", " + M.getPairs().get(ID));
11     }
12 }
13

```

Problems Javadoc Declaration Console Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2018)

```

1, 14
2, 17
3, 12
4, 11
5, 15
6, 12
7, 14
8, 16
9, 13
10, 11

```

**Figure 5 .** Using getChildren() and getPairs() for displaying the results.

Now that we know how to get our results, this can be best demonstrated by two examples. We will read from two different files .txt files: ‘data03.txt’ and ‘data04.txt’ and display the result on the screen.

File *data03.txt* contains following data:

```

C, 0, LeA, 7, 3, 8, 11, 1
C, 1, JoA, 4, 19, 18, 8, 5
C, 2, StA, 8, 7, 9, 17, 1
C, 3, JeB, 13, 16, 11, 12, 9
C, 4, McB, 10, 8, 3, 17, 12
C, 5, MoC, 10, 9, 7, 18, 11
C, 6, DaC, 12, 7, 3, 1, 4
C, 7, BeD, 15, 16, 12, 1, 13
C, 8, CaD, 12, 3, 6, 15, 11
C, 9, DaF, 8, 2, 10, 4, 1
C, 10, AsF, 15, 8, 10, 3, 20
C, 11, HuF, 9, 15, 10, 2, 5
C, 12, HaH, 1, 3, 0, 11, 8
C, 13, JoJ, 14, 11, 1, 13, 9

```

C, 14, KaM, 4, 19, 18, 13, 3  
 C, 15, DrM, 13, 7, 11, 1, 15  
 C, 16, ChM, 3, 6, 7, 9, 13  
 C, 17, CrM, 11, 13, 7, 10, 3  
 C, 18, JoM, 15, 20, 9, 3, 10  
 C, 19, NiS, 5, 6, 18, 17, 3  
 C, 20, TiS, 12, 1, 7, 15, 9  
 C, 21, LiS, 12, 7, 11, 19, 17  
 C, 22, JuU, 6, 19, 7, 0, 13  
 C, 23, JuV, 4, 5, 12, 10, 18  
 C, 24, BrW, 8, 9, 3, 5, 15  
 C, 25, KaX, 9, 1, 13, 14, 20

P, 0, DaM, 1, 10, 18, 1, 22, 2  
 P, 1, BrP, 2, 20, 12, 10, 2, 6  
 P, 2, LaH, 1, 9, 10, 18, 12, 1  
 P, 3, SoF, 1, 4, 16, 0, 6, 11  
 P, 4, EmL, 1, 14, 19, 22, 1, 9  
 P, 5, AlE, 2, 19, 23, 7, 21, 1  
 P, 6, GrG, 1, 22, 16, 8, 3, 23  
 P, 7, KaW, 1, 0, 2, 17, 6, 15  
 P, 8, NiR, 1, 24, 17, 5, 10, 14  
 P, 9, HaR, 1, 11, 5, 2, 18, 24  
 P, 10, JoB, 2, 8, 4, 18, 7, 23  
 P, 11, AmD, 1, 17, 22, 5, 13, 0  
 P, 12, StK, 1, 21, 6, 20, 23, 22  
 P, 13, DeB, 1, 3, 15, 16, 18, 11  
 P, 14, BeH, 2, 13, 16, 5, 12, 21  
 P, 15, RoC, 1, 10, 18, 7, 12, 20  
 P, 16, MiP, 1, 3, 5, 7, 18, 20  
 P, 17, DaCh, 1, 23, 11, 1, 4, 2  
 P, 18, ReW, 1, 5, 23, 22, 21, 0  
 P, 19, ReH, 1, 14, 13, 16, 20, 19  
 P, 20, AnL, 2, 18, 24, 25, 0, 17

When this input is run through the algorithm, the result can be seen in Figure 6 below.

```
1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data03.txt"));
8         M.fraternalPairing();
9         M.printPairs();
10    }
11 }
12
```

Problems Javadoc Declaration Console Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2018)

```
0, 7
1, 19
2, 17
3, 13
4, 10
5, 10
6, 0
7, 16
8, 18
9, 2
10, 15
11, 9
12, 1
13, 14
14, 4
15, 20
16, 3
17, 11
18, 20
19, 5
20, 1
21, 12
22, 6
23, 5
24, 8
25, 14
```

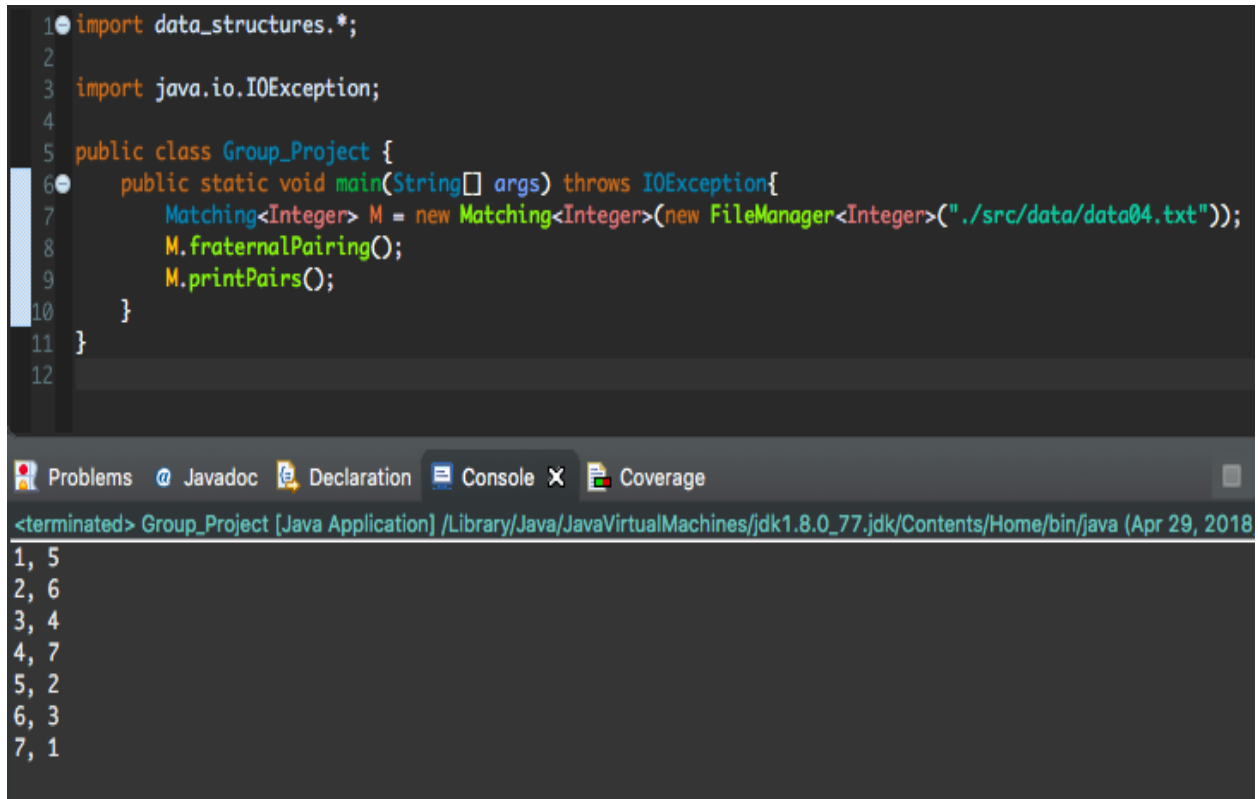
**Figure 6 .** Reading data from data03.txt and producing the pairs and displaying them.

The second example is as follows:

File *data04.txt* contains following data:

```
C, 1, M1, 5, 7, 6, 2, 3, 4, 1
C, 2, M2, 6, 2, 5, 1, 3, 7, 4
C, 3, M3, 4, 3, 7, 2, 1, 5, 6
C, 4, M4, 7, 1, 2, 6, 3, 4, 5
C, 5, M5, 5, 2, 3, 4, 6, 1, 7
C, 6, M6, 3, 4, 5, 6, 7, 1, 2
C, 7, M7, 2, 1, 4, 7, 2, 3, 6
```

P, 1, W1, 1, 5, 3, 4, 6, 7, 1, 2  
 P, 2, W2, 1, 1, 2, 3, 5, 7, 4, 6  
 P, 3, W3, 1, 6, 7, 5, 3, 2, 1, 4  
 P, 4, W4, 1, 6, 5, 7, 2, 1, 4, 3  
 P, 5, W5, 1, 2, 1, 6, 4, 3, 5, 7  
 P, 6, W6, 1, 7, 4, 5, 3, 2, 1, 6  
 P, 7, W7, 1, 3, 4, 7, 5, 6, 2, 1



```

1 import data_structures.*;
2
3 import java.io.IOException;
4
5 public class Group_Project {
6     public static void main(String[] args) throws IOException{
7         Matching<Integer> M = new Matching<Integer>(new FileManager<Integer>("./src/data/data04.txt"));
8         M.fraternalPairing();
9         M.printPairs();
10    }
11 }
12
  
```

Problems Javadoc Declaration Console Coverage

<terminated> Group\_Project [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_77.jdk/Contents/Home/bin/java (Apr 29, 2018)

```

1, 5
2, 6
3, 4
4, 7
5, 2
6, 3
7, 1
  
```

**Figure 7 .** Reading data from data04.txt and producing the pairs and displaying them.

Here, you can find [the code for Matching.java Class.](#)

## Parent Class

This class is consist of quadruple(RollNumber, Name, AvailableChild, and RankedChildren List). It is used to hold all details of only one Parent node.

For example, to in defining a Parent object:

```

Parent<Integer> P1 = new Parent<Integer>();
Parent<Integer> P2 = new Parent<Integer>(P1);
  
```

The second example is shown to just demonstrate how we use another Parent Class (P1) for initializing a new Parent Class (P2).

#### Constructor Summary:

Constructors:	
<code>Parent&lt;T&gt; ()</code>	A default Constructor. Initializes a Parent object so that it represents an empty Child object. For Matching Class this generic type is used as an Integer.
<code>Parent&lt;T&gt;(Parent&lt;T&gt; P)</code>	Copy Constructor. Initializes a Parent object by making a deep copy of the given P.

#### Field and Method Summary:

Fields and Methods	
<code>void</code> <code>setRollNumber(int rollNumber)</code>	Sets a new roll number.
<code>void</code> <code>setName(String name)</code>	Sets a new Name value.
<code>void</code> <code>setAvailChild(int availChild)</code>	Sets a new value for availableChild. This value represents this Parent wants to be paired with how many childrens.
<code>void</code> <code>setRankedChildren(List&lt;Integer&gt; chList)</code>	List of all preferred Children for this Parent.
<code>int</code> <code>getRollNumber()</code>	Returns the RollNumber value.
<code>String</code>	Returns the Name of this Parent.

getName()	
int getAvailChild()	Returns the number of Available Children for this Parent. Zero means this parent does not want any more Child.
List<Integer> getRankedChildren()	List of Ranked Children that this Parent preferred.
int compareTo()	Override method that is used for comparison of this Class.
String toString()	Override method that makes a string from all field values in this Class.

Click here to find [the code for Parent.java Class.](#)

## Child Class

This class is consist of triple(ID, Name, and RankedParent List). This class is used to hold all details of only one Child node. Example of defining a Parent object:

```
Child<Integer> C1 = new Child<Integer>();
```

```
Child<Integer> C2 = new Child<Integer>(P1);
```

The second example is shown to just demonstrate how we use another Child Class (C1) for initializing a new Child Class (C2).

Constructor Summary:

Constructors:	
Child<T> ()	A default Constructor. Initializes a Child object so that it represents an empty Child object. For Matching Class this generic type is used as an Integer.
Child<T>(Child<T> C)	Copy Constructor. Initializes a Child object by making a deep copy of the given C.

Field and Method Summary:

Fields and Methods	
<code>void</code> <code>setId(int iD)</code>	Sets a new ID number.
<code>void</code> <code>setName(String name)</code>	Sets a new Name value.
<code>void</code> <code>setRankedParent(List&lt;Integer&gt; pList)</code>	List of all preferred Parents for this Child.
<code>int</code> <code>getId()</code>	Returns the ID value.
<code>String</code> <code>getName()</code>	Returns the Name of this Child.
<code>List&lt;Integer&gt;</code> <code>getRankedParent()</code>	List of Ranked Parents that this Child preferred.
<code>int</code> <code>compareTo()</code>	Override method that is used for comparison of this Class.
<code>String</code> <code>toString()</code>	Override method that makes a string from all field values in this Class.

Click here to find [the code for Child.java Class.](#)

## FileManager Class

This Class reads from (\*.txt, \*.cvs) files with prepared format as we discussed earlier in [Input Data section](#) for Matching Class and returns, the following:

1. List of Parents' Roll Numbers.
2. List of Children's ID Numbers.



3. HashMap of Parents containing their RollNumber as the Key and all details of that Parent Class as the Value.
4. HashMap of Children containing their ID Number as the Key and all details of that Child Class as the Value.

Here is an example of Defining the FileManager object:

```
FileManager<Integer> F1 = new FileManager<Integer>();
FileManager<Integer> F2 = new FileManager<Integer>('data.txt');
```

The first example demonstrates defining an empty FileManager object and the Second example shows how we define a FileManager object and filling all fields with data from the given file.

#### Constructor Summary:

Constructors:	
<code>FileManager&lt;T&gt; ()</code>	A default Constructor. Initializes a FileManager object so that it represents an empty object. For Matching Class this generic type is used as an Integer.
<code>FileManager&lt;T&gt;(String filePath)</code>	Reads from the given file and Initializes a FileManager object with all data from the file.

#### Field and Method Summary:

Fields and Methods	
<code>String</code> <code>getFilePth()</code>	Returns the given file path.
<code>String[]</code> <code>getLines()</code>	Returns all lines from the file in a String Array. Holds each line of the file in one row of the Array.
<code>List&lt;Integer&gt;</code> <code>getParents()</code>	Returns a List of Parents' Roll Numbers.
<code>List&lt;Integer&gt;</code>	Returns a List of Children's' Roll Numbers.

getChildren()	
<a href="#">HashMap&lt;Integer, Parent&lt;T&gt;&gt;</a> getParentMap()	Returns a Map of all Parents. Uses their Roll Number as the Key and the whole Parent Class as the Value.
<a href="#">HashMap&lt;Integer, Child&lt;T&gt;&gt;</a> getChildrenMap()	Returns a Map of all Children. Uses their IDI Number as the Key and the whole Child Class as the Value.

The Code for Matching Class:

[Matching.java](#)

The Code for FileManager Class:

[FileManager.java](#)

The Code for Parent Class:

[Parent.java](#)

The Code for Child Class:

[Child.java](#)