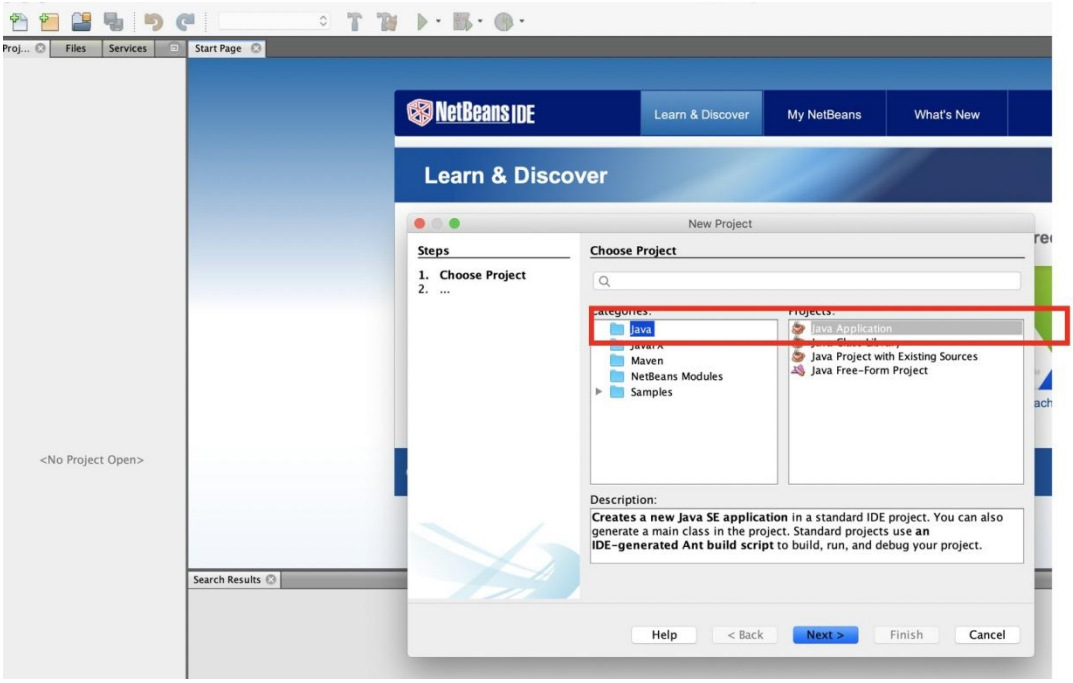
Mini Project on Swing

Requirements

- One class, one file. Don't create multiple classes in the same .java file
- Don't use static variables and methods
- Encapsulation: make sure you protect your class variables and provide access to them through get and
- set methods
- all the classes are required to have a constructor that receives **all** the attributes as parameters and
- updates the attributes accordingly
- all the classes are required to have an "empty" constructor that receives **no** parameters but updates all
- the attributes as needed
- Follow **Horstmann's Java Language Coding Guidelines**
- Organized in packages (MVC - Model - View Controller)

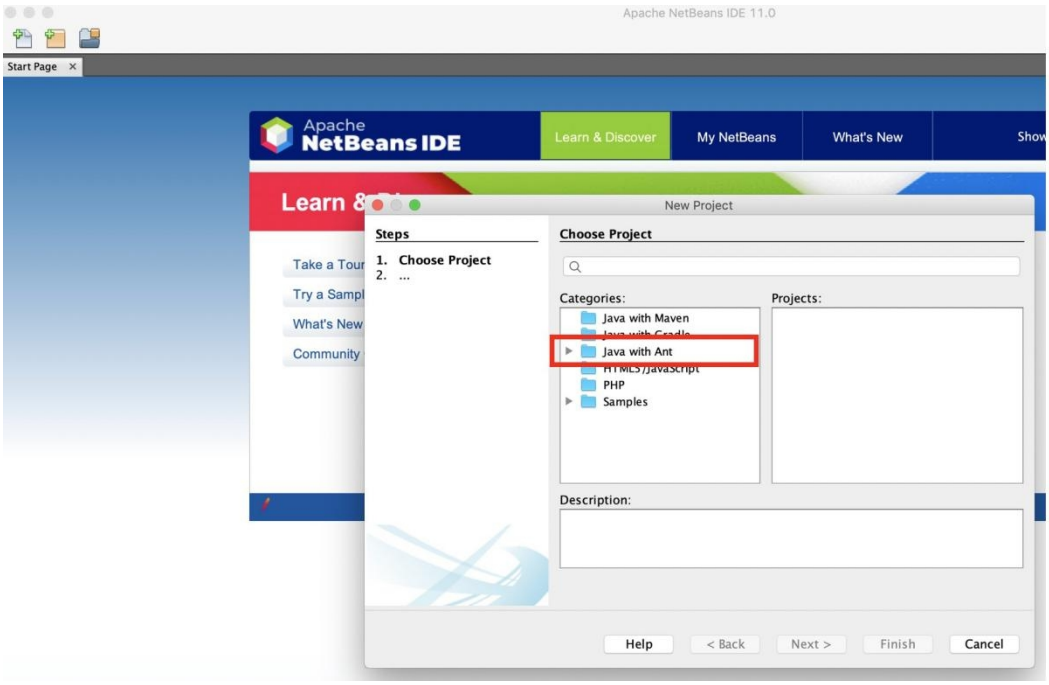
Creating aProject

You will create a Java project (Netbeans 8.2 or earlier) or a Java project with Ant (Netbeans 9.0 or later).



In Netbeans 8.2 or earlier, you will create a new Java project and a Java application

Mini Project on Swing

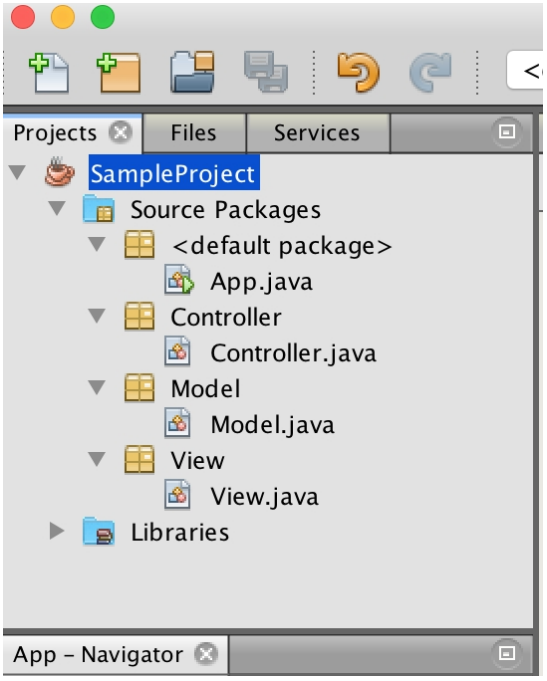


In Netbeans 9 or later, you will create a new Java with Ant project

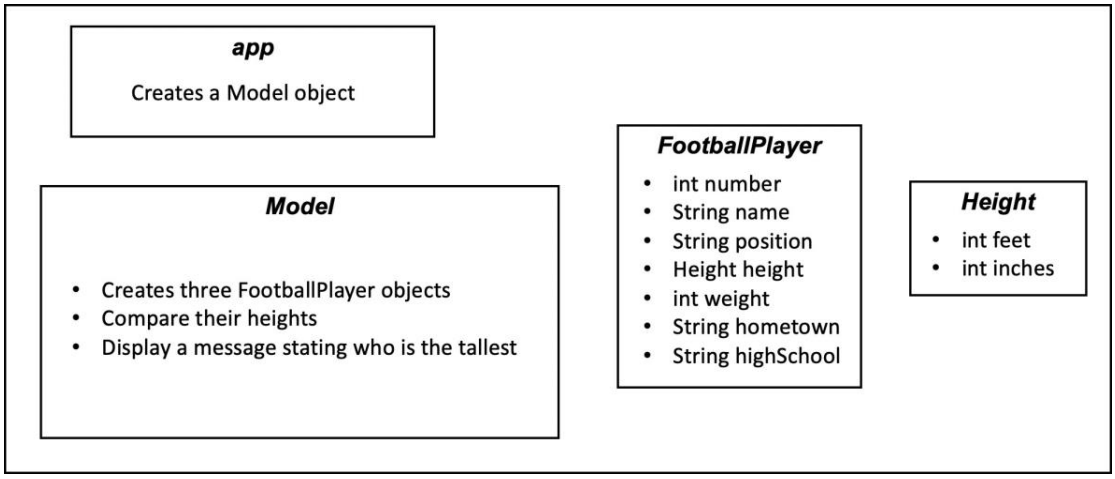
All the submissions have to follow the MVC (Model-View-Controller) package structure.

Your Project

- <default package>
  - App.java
- Controller
  - ... list of controller classes
- Model
  - ... list of Model classes
- View
  - …list of View classes



Mini Project on Swing



Start and create one with

- <default package>
  - App.java
- Model
  - Model.java
  - FootballPlayer.java
  - Height.java

Functionality

- The application App creates a Model object
- The Model class
  - creates 3 FootballPlayer objects
  - compares the height of the three FootballPlayer objects
  - displays a message stating who is the tallest

The classes

App

- it has the main method which is the method that Java looks for and runs to start any application
- it creates an object (an instance) of the Model class

Model

- this is the class where all the action is going to happen
- it creates three football players and compare their heights and displays a message stating who is taller or if they are of the same height
- make sure you test your application for all the possible cases

FootballPlayer

has the following attributes

- int number;
- String name;
- String position;
- Height height;
- int weight;
- String hometown;

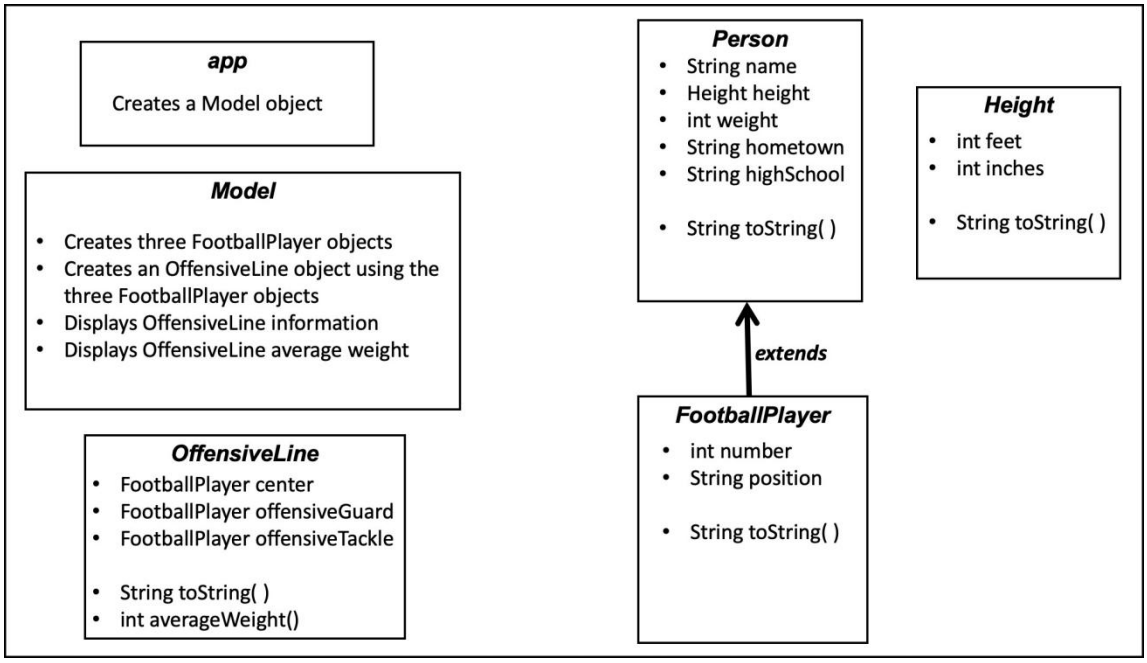
- String highSchool;

Height

- it is a class (or type) which is used in FootballPlayer defining the type of the attribute height
- it has two attributes
- int feet;
- int inches
- it also has a method that returns a formatted string with feet and inches
- for instance: 5'2"

Mini Project on Swing

Part II:



Create a Netbeans project with

- <default package>
  - App.java
- Model
  - Model.java
  - FootballPlayer.java
  - Height.java
  - **Person.java**
  - **OffensiveLine.java**

Functionality

- The application App creates a Model object that
- The Model class
  - creates 3 FootballPlayer objects
  - creates an Offensive object using the 3 FootballPlayer objects
  - displays information about the OffensiveLine object and its 3 players
    - it is a **requirement** that this should be done using the **toString( )** method in OffensiveLine, which will use **toString( )** in FootballPlayer
  - display the average weight of the OffensiveLine
    - this will be done using the averageWeight in the OffensiveLine

The classes

App

- it has the main method which is the method that Java looks for and runs to start any application
- it creates an object (an instance) of the Model class

Mini Project on Swing

Model

- this is the class where all the action is going to happen
- it creates three football players
- It creates an OffensiveLine object using the three players
- displays information about the OffensiveLine
  - this has to be done using the OffensiveLine object
  - this is really information about its 3 players
  - the format is free as long as it contains all the information about each of the 3 player
- displays the average weight of the OffensiveLine
  - this has to be done using the OffensiveLine object
  - this has to call the averageWeight method in OffensiveLine

Person

has the following attributes

- String name;
- Height height;
- int weight;
- String hometown;
- String highSchool

and a method

- String toString( )
  - toString() overrides the superclass Object toString() method
  - toString() returns information about this class attributes as a String

encapsulation

- if you want other classes in the same package yo have access to the attributes, you need to make them protected instead of private.

FootballPlayer

has the following attributes

- int number;
- String position;

and a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String

Height

it is a class (or type) which is used in FootballPlayer defining the type of the attribute height
it has two attributes

- int feet;
- int inches

it also has a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String
  - it returns a formatted string with feet and inches
  - for instance: 5'2"

Mini Project on Swing

OffensiveLine
has the following attributes

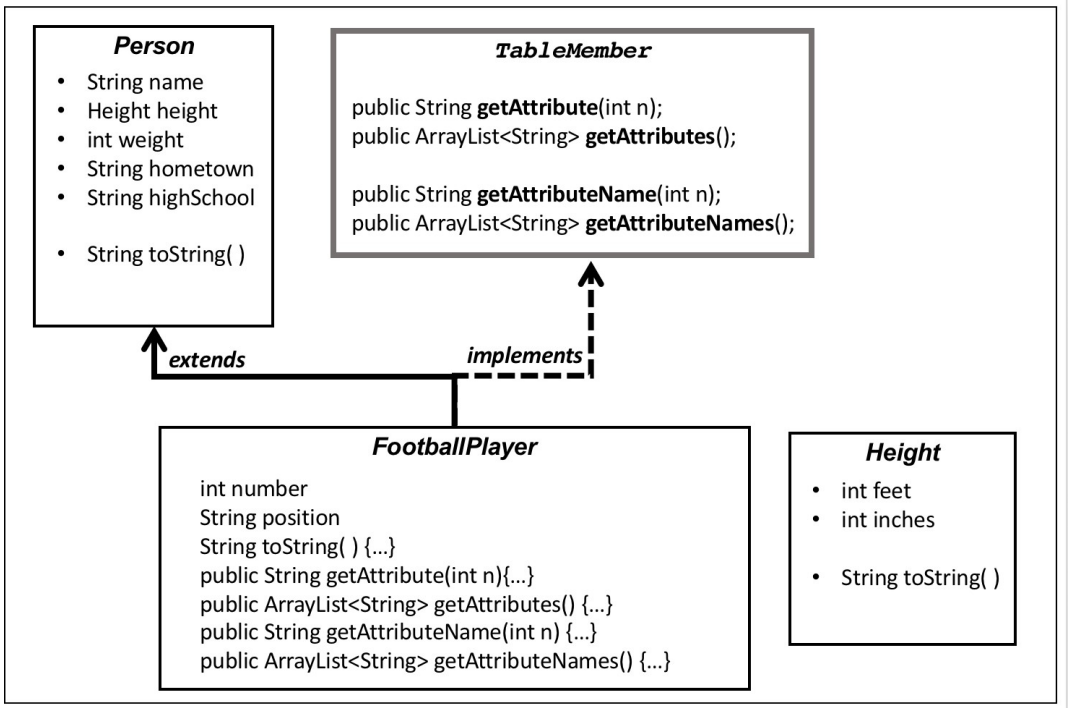
- FootballPlayer center
- FootballPlayer offensiveGuard
- FootballPlayer offensiveTackle
- They might also be stored in an ArrayList

and two methods

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about the 3 players attributes as a String
- int averageWeight()
  - calculates and returns the average weight of the OffensiveLine.
  - it is calculated based on the weight of each of its players

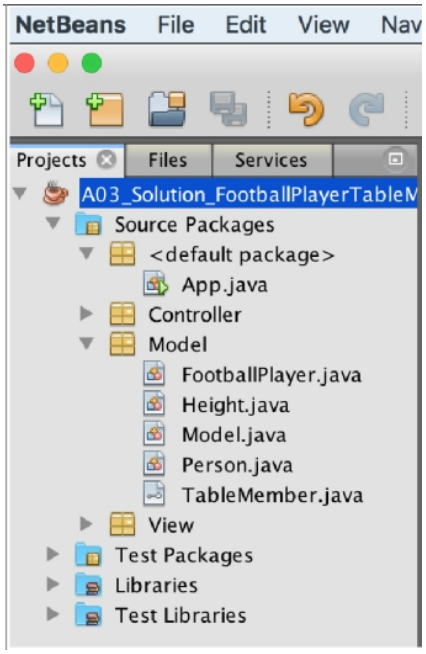
Mini Project on Swing

Part III:



Create a Netbeans project with

- <default package>
  - App.java
- Model
  - FootballPlayer.java
  - Height.java
  - Model.java
  - Person.java
  - TableMember.java
- Controller (not being used)
- View (not being used)



Functionality

- The application App creates a Model object that
- The Model class
  - creates one FootballPlayer object
  - displays information to test the newly implemented methods

Mini Project on Swing

The classes

App

- it has the main method which is the method that Java looks for and runs to start any application
- it creates an object (an instance) of the Model class
- has to have the code segment

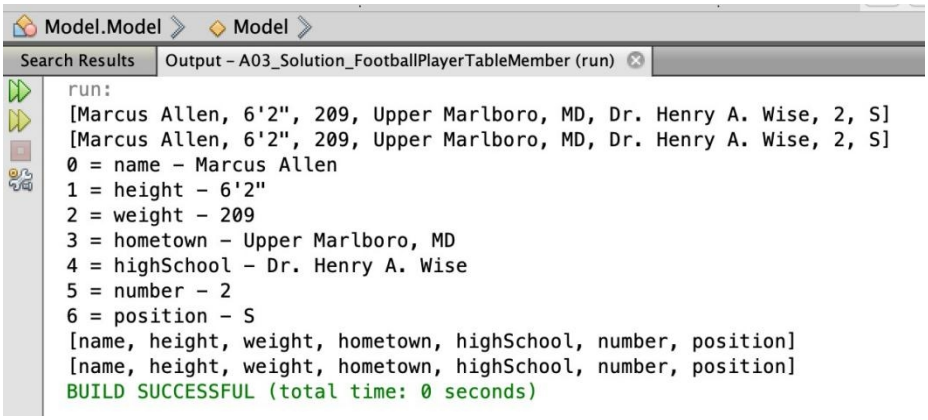
```
Model model = new Model();
```

Model

- has to have the code below

```
package Model; public class Model
{
public Model()
{
FootballPlayer fp = new FootballPlayer(2, "Marcus Allen", "S", 6, 2, 209, "Upper
Marlboro, MD", "Dr. Henry A. Wise");
System.out.println(fp.getAttributes().toString());
System.out.println(fp.getAttributes().toString()); //Yes, we are running
getAttributes twice jus t to test it
for (int i = 0; i < fp.getAttributes().size(); i++)
{
System.out.println(i + " = " + fp.getAttributeName(i) + " - " +
fp.getAttribute(i));
}
System.out.println(fp.getAttributeNames().toString());
System.out.println(fp.getAttributeNames().toString()); //Yes, we are running
getAttributeNames twice just to test it
//if the implementation of TableMember by FootballPlayer is correct,
//the output will be
//
//[Marcus Allen, 6'2", 209, Upper Marlboro, MD, Dr. Henry A. Wise, 2, S]
//[Marcus Allen, 6'2", 209, Upper Marlboro, MD, Dr. Henry A. Wise, 2, S]
//0 = name - Marcus Allen
//1 = height - 6'2"
//2 = weight - 209
//3 = hometown - Upper Marlboro, MD
//4 = highSchool - Dr. Henry A. Wise
//5 = number - 2
//6 = position - S
//[name, height, weight, hometown, highSchool, number, position]
//[name, height, weight, hometown, highSchool, number, position]  //
}
}
```

sample output of running Model.java



Mini Project on Swing

- it creates a FootballPlayer object
- uses this object to test and display the result of the newly implemented methods
  - getAttribute(int n)
  - getAttributes( )
  - getAttributeName(int n)
  - getAttributeNames( )

Person

has the following attributes

- String name;
- Height height;
- int weight;
- String hometown;
- String highSchool

and a method

- String toString( )
  - toString() overrides the superclass Object toString() method
  - toString() returns information about this class attributes as a String

encapsulation

- if you want other classes in the same package yo have access to the attributes, you need to make them protected instead of private.

FootballPlayer

has the following attributes

- int number;
- String position;

has a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String

and the methods coming from the interface TableMember getAttribute(int n)

  - getAttributes( )
  - getAttributeName(int n)
  - getAttributeNames( )

Height

it is a class (or type) which is used in FootballPlayer defining the type of the attribute height
it has two attributes

- int feet;
- int inches

it also has a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String
  - it returns a formatted string with feet and inches
  - for instance: 5'2"

Mini Project on Swing

The interface

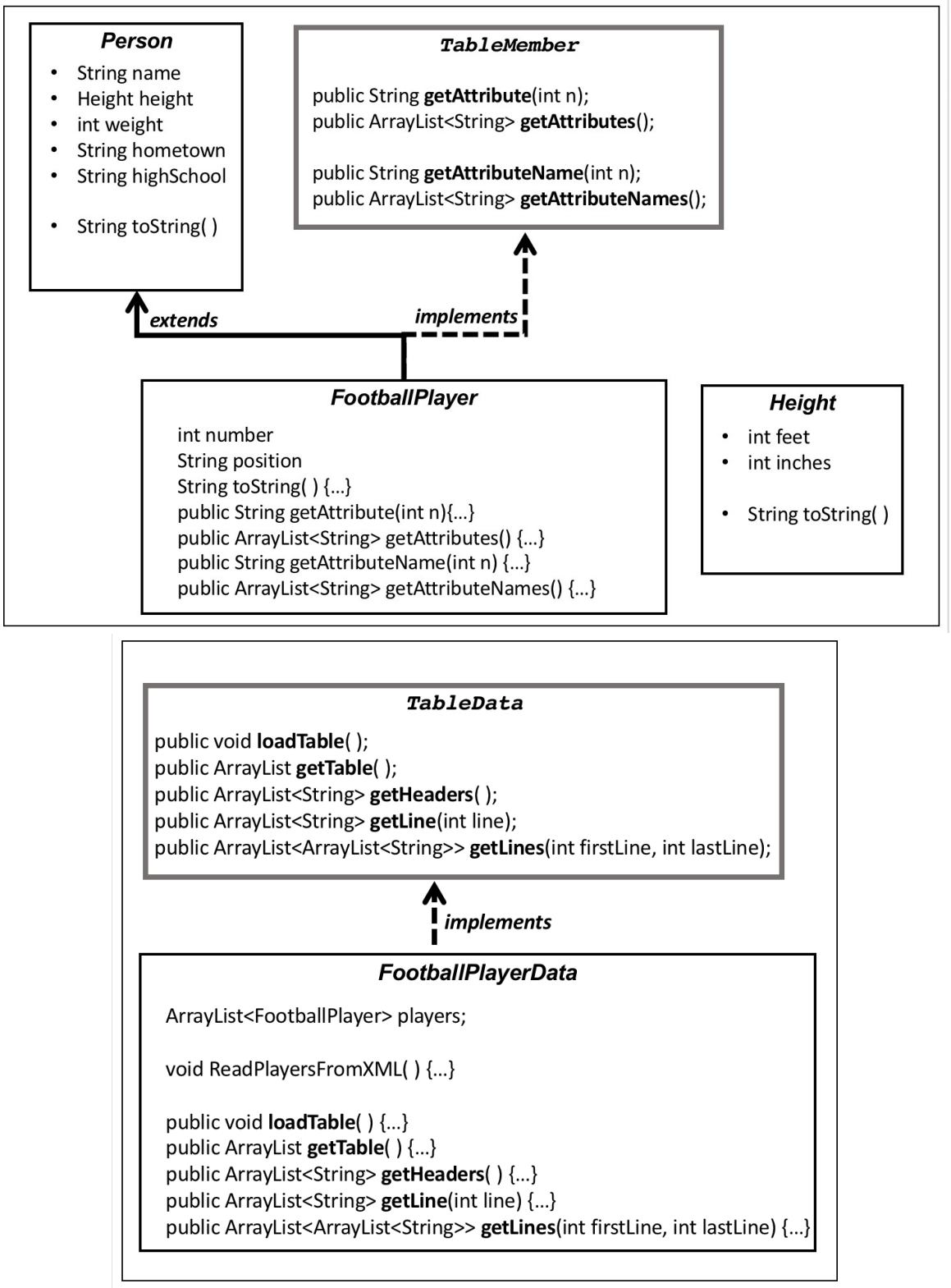
TableMember

These are the four methods that FootballPlayer has to implement

- public String getAttribute(int n);
  - Returns the value of a specific attribute. The input parameter start with 0 for the first attr ibute, then 1 for the second attribute and so on.
- public ArrayList<String> getAttributes();
  - Returns the value of all attributes as an ArrayList of Strings.
- public String getAttributeName(int n);
  - Returns the name of a specific attribute. The input parameter start with 0 for the first attribute, then 1 for the second attribute and so on.
- public ArrayList<String> getAttributeNames();
  - Returns the name of all attributes as an ArrayList of Strings.

Mini Project on Swing

Part IV:



FootballPlayerData will implement the interface **TableData** writing real code for all the abstract methods. **FootballPlayer** will implement the interface **TableMember** writing real code for all the abstract methods.

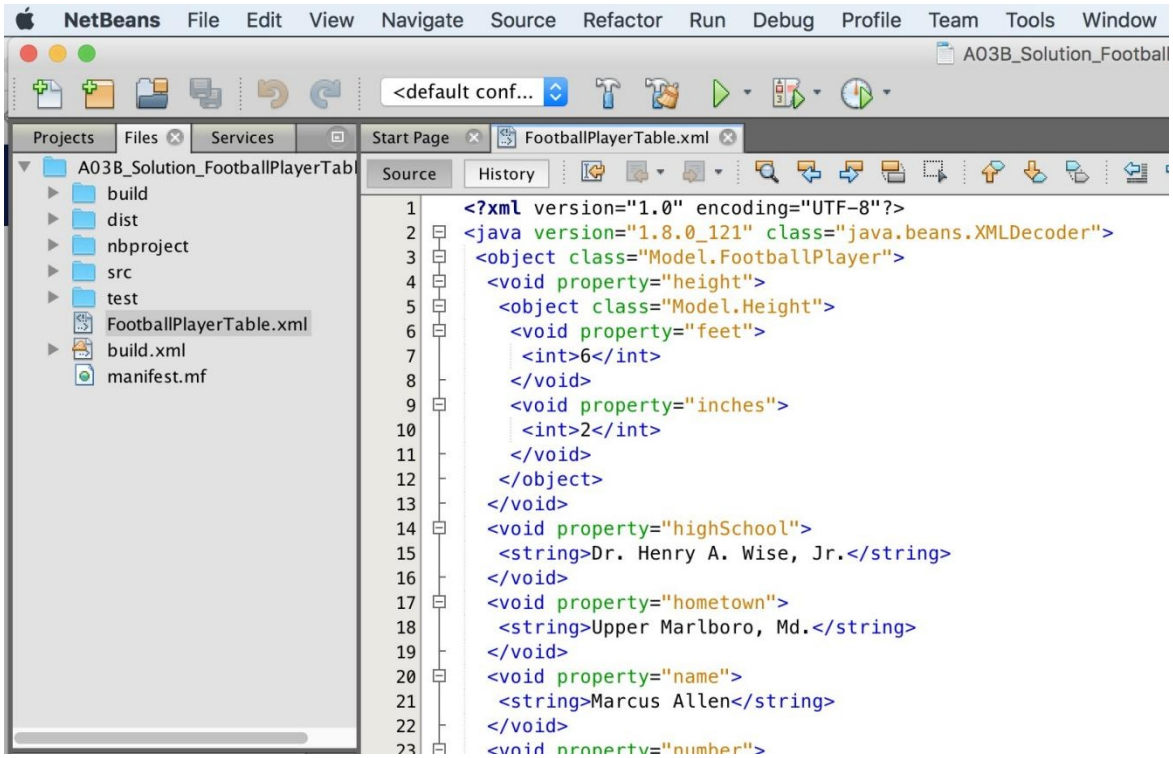
Mini Project on Swing

```
public class Model
{
    private FootballPlayerData fpData = new FootballPlayerData();
    public Model()
    {
        System.out.println("============================================================");
        System.out.println("the table size is "+ fpData.getTable().size());
        System.out.println("============================================================");
        this.displayArray(fpData.getHeaders());
        this.displayArray(fpData.getLine(4));
        System.out.println("============================================================");
        this.displayArray(fpData.getHeaders());
        this.displayArray(fpData.getLine(121));
        System.out.println("============================================================");
        this.displayArrayOfArrays(fpData.getLines(70, 72));
        System.out.println("============================================================");
        this.displayArrayOfArrays(fpData.getLines(100, 101));
        System.out.println("============================================================");
        System.out.println("the table size is "+ fpData.getTable().size());
        System.out.println("============================================================");
    }
```

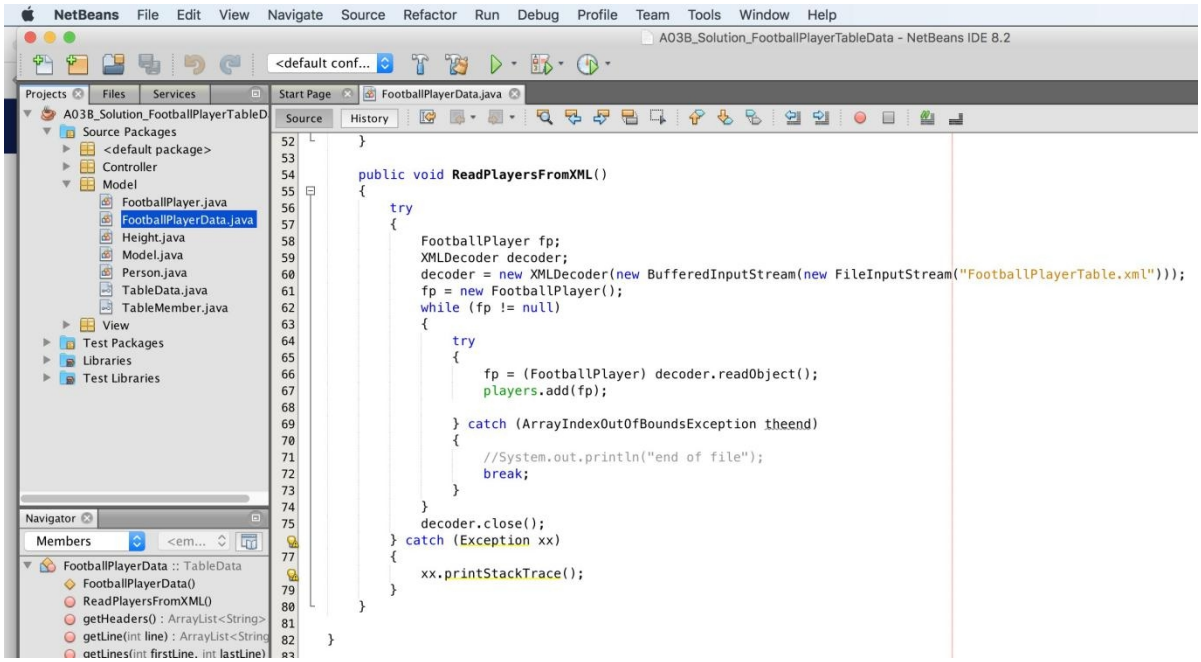
Model is used to test if the **TableData** interface was implemented correctly by **FootballPlayerData**.

FootballPlayerTable.xml

- a XML file with all the FootballPlayer objects coded in XML. The method ReadPlayersFromXML( ) (which is ready) reads this file and create the FootballPlayer objects that will be loaded in the ArrayList.
- The method ReadPlayersFromXML( ) requires that the classes it uses (FootballPlayer, Height, and Person)
  - have an empty constructor (besides the "regular" constructor you are using
  - have all getters and setters (and you should already if you are using the required encapsulation)

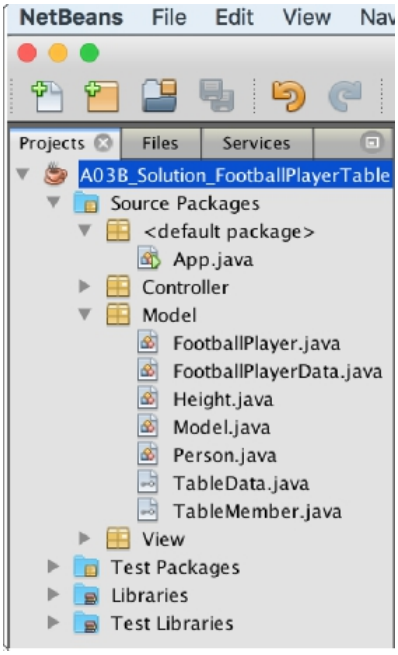


Mini Project on Swing



Create a Netbeans project with

- <default package>
  - App.java
- Model
  - FootballPlayer.java
  - FootballPlayerData.java
  - Height.java
  - Model.java
  - Person.java
  - TableData.java
  - TableMember.java
- Controller (not being used)
- View (not being used)



Functionality

- The application App creates a Model object
- The Model class
  - creates one FootballPlayer object
  - displays information to test the newly implemented methods

The classes

App

- it has the main method which is the method that Java looks for and runs to start any application
- it creates an object (an instance) of the Model class
- has to have the code segment

```
Model model = new Model();
```

Model

- has to have the code below

```
package Model;
import java.util.ArrayList;

public class Model
{
private FootballPlayerData fpData = new FootballPlayerData();

public Model()
{
System.out.println("================================================================");
System.out.println("the table size is "+ fpData.getTable().size());
System.out.println("================================================================");
this.displayArray(fpData.getHeaders());
this.displayArray(fpData.getLine(4));
System.out.println("================================================================");
this.displayArray(fpData.getHeaders()); this.displayArray(fpData.getLine(121));
System.out.println("================================================================");
this.displayArrayOfArrays(fpData.getLines(70, 72));
System.out.println("================================================================");
this.displayArrayOfArrays(fpData.getLines(100, 101));
System.out.println("================================================================");
System.out.println("================================================================");
System.out.println("the table size is "+ fpData.getTable().size());
}

public void displayArrayOfArrays(ArrayList<ArrayList<String>> manyLines)
{
for (int i = 0; i < manyLines.size(); i++)
{
displayArray(manyLines.get(i));
}

}

public void displayArray(ArrayList<String> oneLineOnly)
{
for (int k = 0; k < oneLineOnly.size(); k++)
{
System.out.print(oneLineOnly.get(k)); System.out.print(" ");
}
System.out.println("");

}

/**
* @return the fpData
*/
public FootballPlayerData getFpData()
{
return fpData;
}

/**
* @param fpData the fpData to set
*/
public void setFpData(FootballPlayerData fpData)
{
this.fpData = fpData;
}
}
```

Mini Project on Swing

- output of running Model.java

```
============================================================
the table size is 124
============================================================
number position name height weight hometown highSchool
26 RB Saquon Barkley 5'11" 222 Coplay, Pa. Whitehall
============================================================
number position name height weight hometown highSchool
44 SN Tyler Yazujian 5'11" 264 Royersford, Pa. Spring-Ford
============================================================
7 WR Geno Lewis 6'1" 205 Wilkes-Barre, Pa. Wyoming Valley West
93 P Robby Liebel 6'2" 194 St. Petersburg, Fla. IMG Academy
5 S Jordan Lucas 6'0" 199 New Rochelle, N.Y. New Rochelle
============================================================
62 G Zach Simpson 6'3" 272 Hollidaysburg, Pa. Hollidaysburg Area
24 S Anthony Smith 6'0" 206 Dover, N.J. Pope John XXIII
============================================================
the table size is 124
============================================================
```

- it creates a FootballPlayerData object
- uses this object to test and display the result of the newly implemented methods
  - getHeaders( )
  - getLine(int n)
  - getLines(int firstLine, int lastLine)

Person

has the following attributes

- String name;
- Height height;
- int weight;
- String hometown;
- String highSchool

and a method

- String toString( )
  - toString() overrides the superclass Object toString() method
  - toString() returns information about this class attributes as a String

encapsulation

- if you want other classes in the same package yo have access to the attributes, you need to make them protected instead of private.

FootballPlayer

has the following attributes

- int number;
- String position;

has a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String

and the methods coming from the interface TableMember getAttribute(int n)

  - getAttributes( )
  - getAttributeName(int n)
  - getAttributeNames( )

Mini Project on Swing

Height

it is a class (or type) which is used in FootballPlayer defining the type of the attribute height
it has two attributes

- int feet;
- int inches

it also has a method

- String toString( )
  - toString( ) overrides the superclass Object toString( ) method
  - toString( ) returns information about this class attributes as a String
  - it returns a formatted string with feet and inches
  - for instance: 5'2"

FootballPlayerData

Below is the code for FootballPlayerData

```
public class FootballPlayerData
{
    private ArrayList<FootballPlayer> players;

    public FootballPlayerData()
    {
        players = new ArrayList<>();
    }

    public void ReadPlayersFromXML()
    {
        try
        {
            FootballPlayer fp;
            XMLDecoder decoder;
            decoder = new XMLDecoder(new BufferedInputStream(new FileInputStream("FootballPlayerTable.xml")));
            fp = new FootballPlayer();
            while (fp != null)
            {
                try
                {
                    fp = (FootballPlayer) decoder.readObject();
                    players.add(fp);

                } catch (ArrayIndexOutOfBoundsException theend)
                {
                    //System.out.println("end of file");
                    break;
                }
            }
            decoder.close();
        } catch (Exception xx) {xx.printStackTrace();}
    }
}
```

- has the following attribute
  - ArrayList<FootballPlayer> players;
    - this is the array with all the FootballPlayer objects and the methods coming from the interface Data (these are the methods you need to implement and write the full code for)
- public void loadTable();
- public ArrayList getTable();
- public ArrayList<String> getHeaders();
- public ArrayList<String> getLine(int line);
- public ArrayList<ArrayList<String>> getLines(int firstLine, int lastLine);

- and a method to read the FootballPlayer objects from a XML and load the ArrayList<FootballPlayer> players
  - public void ReadPlayersFromXML( ) this method will be used in loadTable( )

The interface

TableData

These are the five methods that FootballPlayerData has to implement.

- public void loadTable();
  - It needs to be called in the constructor to make the table ready and available right away. It loads real data into the array list.
  - The FootballPlayerData starter class has a method that help with this task.
  - The method ReadPlayersFromXML( ) read the FootballPlayer objects from an XML file and load them into the players ArrayList.
- public ArrayList<FootballPlayer> getTable();
  - Returns the full table of data. Although right now is not required, soon you will need thi s method to return a specific ArrayList instead of a generic ArrayList.
- public ArrayList<String> getHeaders();
  - The method will return an ArrayList of Strings with the names of the FootballPlayer fields (the "headers").
- public ArrayList<String> getLine(int line);
  - The method will return an ArrayList of Strings with the values of the FootballPlayer object in a chosen line "n" in the table.
- public ArrayList<ArrayList<String>> getLines(int firstLine, int lastLine);
  - This methods used the getLine( ) method. It gets a set of lines (each one of them an array of Strings) from getLine(int n) and adds them to an array of arrays. It returns this array of arrays.

- getLine(int line) can use FootballPlayer getAttributes( )
- getHeaders( ) can use FootballPlayer getAttributeNames( )
- getLines(int firstLine, int lastLine) can use FootballPlayerData getLine(int line)

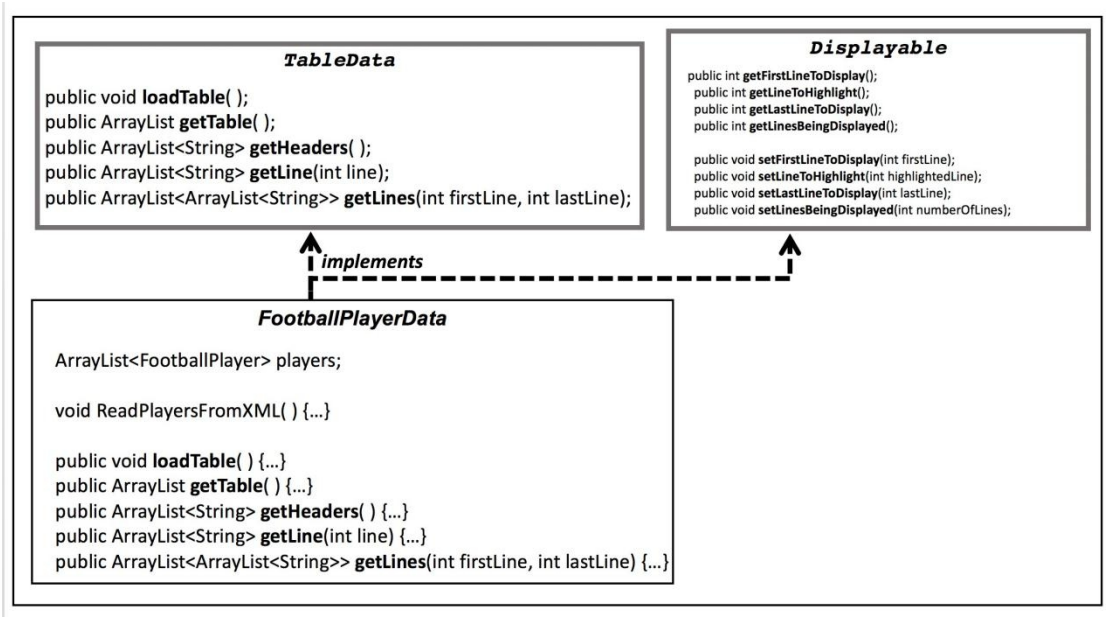
Mini Project on Swing

Part V:

FootballPlayerData

FootballPlayerData will now implement a second interface, called displayable.

```
public class FootballPlayerData implements TableData, Displayable
{
```



Graphics

IST 242 - Table Viewer

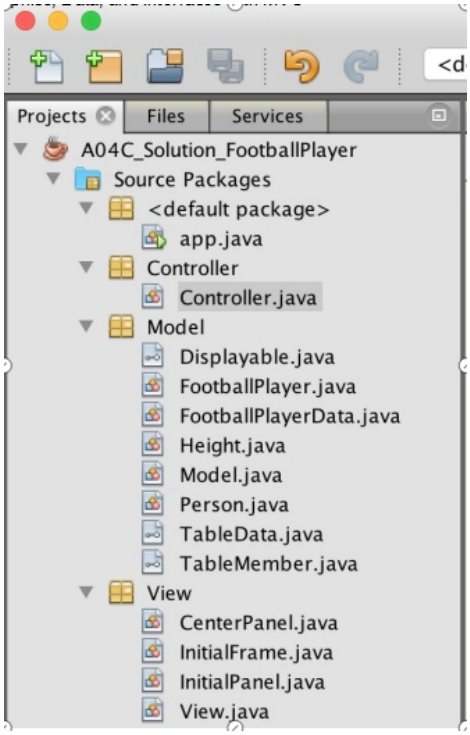
number	position	name	height	weight	hometown	highSchool
2	S	Marcus Allen	6'2"	209	Upper Marlboro, Md.	Dr. Henry A. Wise, Jr.
37	CB	Kyle Alston	5'9"	180	Robbinsville, N.J.	Robbinsville
28	S	Troy Apke	6'1"	198	Mt. Lebanon, Pa.	Mount Lebanon
35	LB	Matthew Baney	6'0"	225	State College, Pa.	State College
26	RB	Saquon Barkley	5'11"	222	Coplay, Pa.	Whitehall
91	DT	Tarow Barney	6'1"	306	Bainbridge, Ga.	Bainbridge
52	G/C	Ryan Bates	6'4"	284	Warrington, Pa.	Archbishop Wood
60	T	Noah Beh	6'6"	294	Moscow, Pa.	Scranton Prep
11	LB	Brandon Bell	6'1"	231	Mays Landing, N.J.	Oakcrest
89	WR	Gordon Bentley	6'2"	201	Ambler, Pa.	Wissahickon
32	S	Joe Berg	6'0"	197	Mundelein, Ill.	Carmel Catholic
13	WR	Saeed Blacknall	6'3"	211	Manalapan, N.J.	Manalapan
91	PK	Nick Boumerhi	5'9"	173	Phillipsburg, Pa.	Phillipsburg-Osceola
43	LB	Manny Bowen	6'1"	200	Barnegat, N.J.	Barnegat
83	TE/H	Nick Bowers	6'4"	255	Kittanning, Pa.	Kittanning Senior
81	TE/H	Adam Breneman	6'4"	245	Mechanicsburg, Pa.	Cedar Cliff
75	T	Brendan Brosnan	6'6"	297	Park Ridge, Ill.	Maine South
19	DE	Torrence Brown	6'3"	250	Tuscaloosa, Ala.	Tuscaloosa Academy
97	DE	Ryan Buchholz	6'6"	254	Malvern, Pa.	Great Valley
40	LB	Jason Cabinda	6'1"	245	Flemington, N.J.	Hunterdon Central

Mini Project on Swing

You will use JLabels and/or JButtons for the basic units of display.

Create a Netbeans project with

- <default package>
  - App.java
- Model
  - Displayable.java
  - FootballPlayer.java
  - FootballPlayerData.java
  - Height.java
  - Model.java
  - Person.java
  - TableData.java
  - TableMember.java
- Controller
  - Controller.java
- View
  - CenterPanel.java
  - InitialFrame.java
  - InitialPanel.java
  - View.java



The classes

All these classes belong to View.

- View creates the InitialPanel, the first and enclosing pane
- InitialPanel has one other panel (in future assignments it will have more panels)

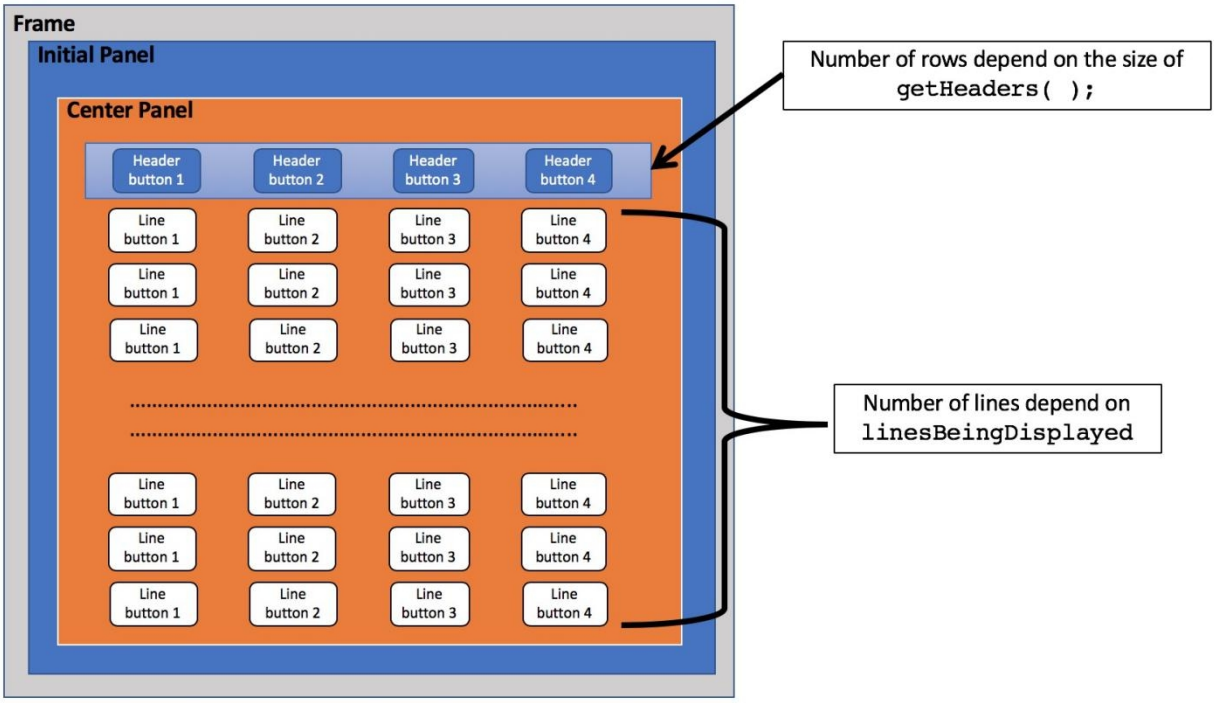
CenterPanel

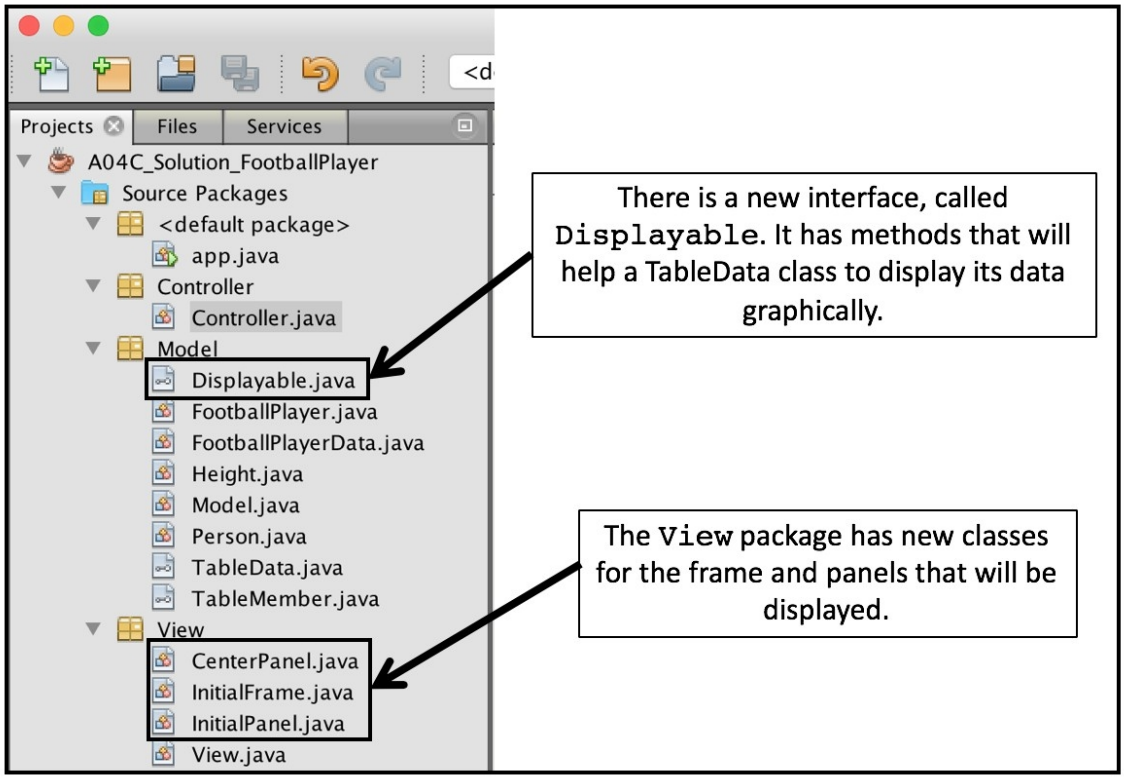
- with a row of buttons with the headers from the table each attribute name in its own button
- the number of buttons depend on the size of the array returned by getHeaders( )
- with a row of buttons with the lines from the table each attribute value in its own button
- the number of lines depend on the value of the attribute linesBeingDisplayed

CenterPanel

- it is the panel in charge of displaying the data
- it has a row of buttons or labels to display the headers. This row should be graphically distinct from the lines below it. This can be done by using labels, and/or by using a different background and/or a different font.
- it has also a number of lines to display the table data. These lines should be graphically distinct from the headers
- it needs a layout, most likely a GridLayout

Mini Project on Swing





App

- it has the main method which is the method that Java looks for and runs to start any application
- it creates an object (an instance) of the Model class, an object (an instance) of the View class and an object (an instance) of the Controller class.
- the suggested initial code in app

Mini Project on Swing

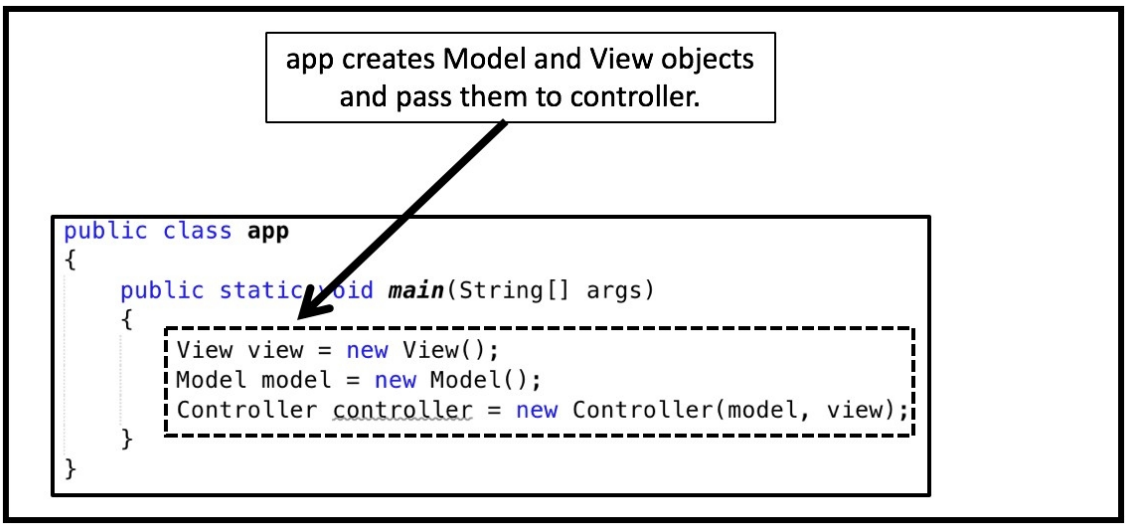
```
public class app
{
    public static void main(String[] args)
    {
        View view = new View();
        Model model = new Model();
        Controller controller = new Controller(model, view);
    }
}
```

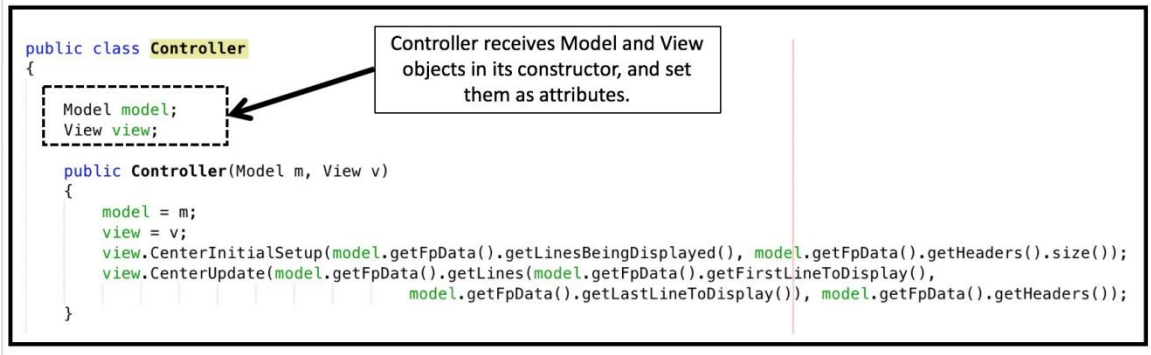
Controller

- the suggested initial code in Controller

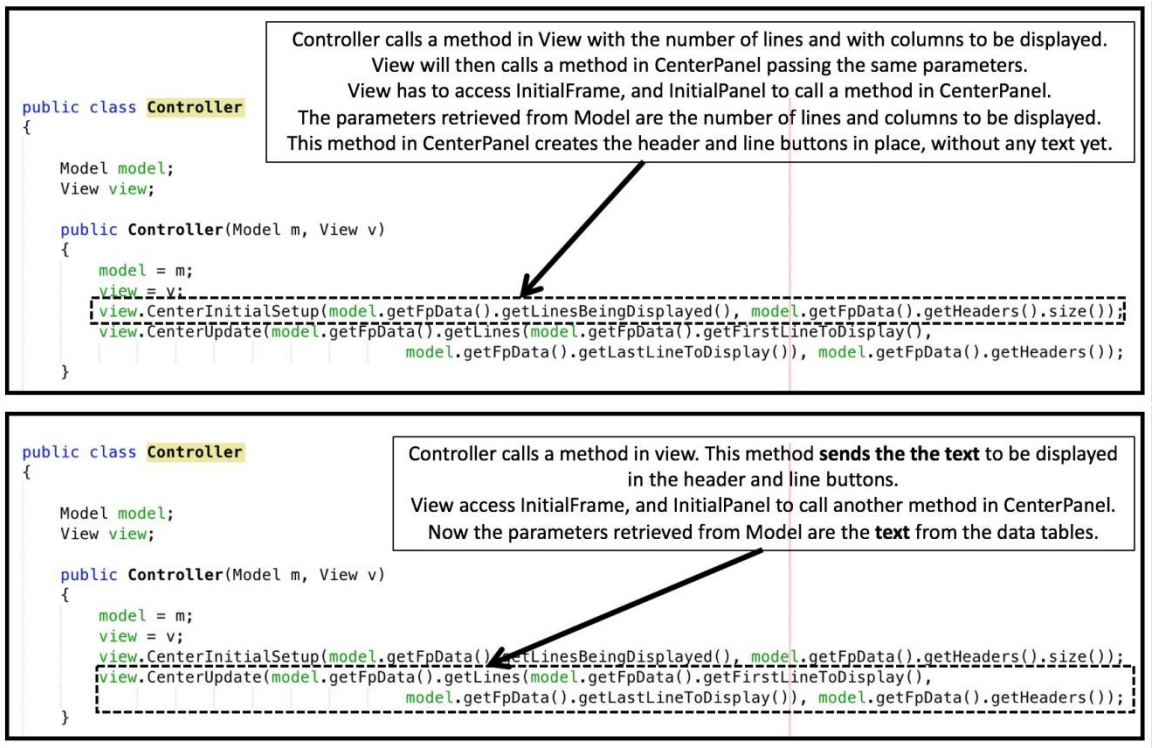
```
public class Controller
{
    Model model;
    View view;

    public Controller(Model m, View v)
    {
        model = m;
        view = v;
        view.CenterInitialSetup(model.getFpData().getLinesBeingDisplay
ed(), model.getFpData().getHeaders().size());
        view.CenterUpdate(model.getFpData().getLines(model.getFpData().
getFirstLineToDisplay(), model.getFpData().getLastLineToDisplay()),
model.getFpData().getHeaders());
    }
}
```





Mini Project on Swing



Displayable

- public int getFirstLineToDisplay();
- public void setFirstLineToDisplay(int firstLine);
  - these first two methods are about an int attribute that will hold the number of the first line to be displayed. The number represents the index of an element in the array of the class that implements the TableData interface.
- public int getLineToHighlight();
- public void setLineToHighlight(int highlightedLine);
  - the two methods above are about an int attribute that will hold the number of the line on the screen that should be highlighted. It will be used only in a later assignment but it is part of the interface and needs to be implemented even if it is not fully functional yet.
- public int getLastLineToDisplay();
- public void setLastLineToDisplay(int lastLine);
  - these two methods are about an int attribute that will hold the number of the last line to be displayed. The number represents the index of an element in the array of the class that implements the TableData interface.
- public int getLinesBeingDisplayed();
- public void setLinesBeingDisplayed(int numberOfLines);
  - these two methods are about an int attribute that will hold the number of the lines that will appear on the screen at one time. It will be most likely 20 but it should a variable. The application should work with any number of lines. So if this number is set to 10 or 15, this is the number of lines that should appear on the screen.

- the suggested initial code in Displayable

```
package Model;

public interface Displayable
{
    public int getFirstLineToDisplay();
    public int getLineToHighlight();
    public int getLastLineToDisplay();
    public int getLinesBeingDisplayed();
    public void setFirstLineToDisplay(int firstLine);
    public void setLineToHighlight(int highlightedLine);
    public void setLastLineToDisplay(int lastLine);
    public void setLinesBeingDisplayed(int numberOfLines);
}
```

Updating Graphics

- If you make changes in the layout or add and remove one or many components at once, then you might force the Panel to be updated using the method validate().
- validate( ) recalculates the layout after some changes are made. You may also need to repaint().
- repaint( ) forces the screen to be refreshed.

For instance,

```
add(p1);
add(p2);
...
remove(p1);
remove (p2);
add(p3);
add(p4);
validate();
repaint();
```

Arrays of JButtons

Keep in mind that graphics components are classes and objects just like any other class and object in Java. The only difference is that they have the ability to have a graphical display.

Then, yes, you can have an array of JButtons or JPanels for instance.

```
ArrayList<JButton> buttonArray; // declares the JButton ArrayList
ArrayList<JLabel> labelArray; // declares an ArrayList of JLabels
```

Mini Project on Swing

Part VI:

Functionality

- Implement Listener Interfaces
  - MouseWheelListener
  - ActionListener
- Add a MouseWheelListener to enable scrolling
- Add an ActionListener to enable listening of specific buttons
- The scrolling functionality will be implemented with the Interface MouseWheelListener

```
public class Controller
{

    Model model;
    View view;

    public Controller(Model m, View v)
    {
        model = m;
        view = v;
        view.CenterInitialSetup(model.getFpData().getLinesBeingDisplayed(), model.getFpData().getHeaders().size());
        model.getFpData().setFirstLineToDisplay(0);
        view.CenterUpdate(model.getFpData().getLines(model.getFpData().getFirstLineToDisplay(), model.getFpData().ge
        addScrolling();
    }

    private void addScrolling()
    {
        view.getInitialframe().getInitialPanel().getCp().addMouseWheelListener(
                new MouseWheelListener()
        {
            @Override
            public void mouseWheelMoved(MouseWheelEvent e)
            {
                int units = e.getUnitsToScroll();
                //code to make the table scroll
                //.....
            }

        }
        );
        //.......
```

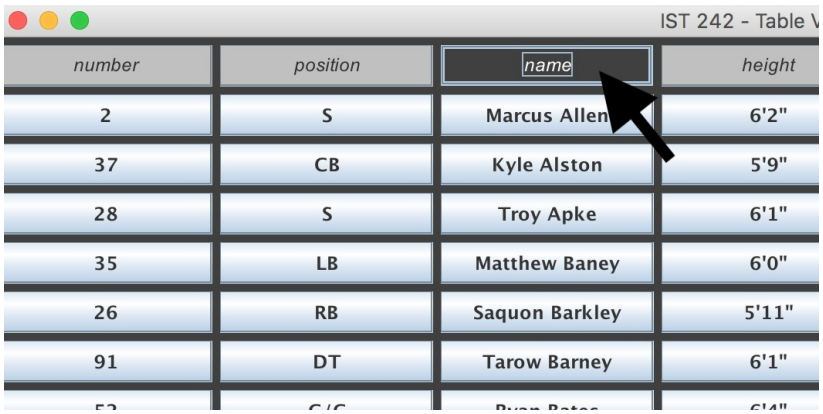
- You cannot use a JScrollPane or any other embedded Java scrolling functionality.
- You need to write the scrolling functionality yourself using MouseWheelListener implemented as an inner listener.
- Strategies to implement scrolling
  1. keep sending a number of lines as Strings at a time to the panel. Which lines to be sent will depend on the mouse wheel movement.
  2. If you need to define the first line to display, use the method setFirstLineToDisplay(int n)
  3. If you need to retrieve the first line to display, use the method getFirstLineToDisplay()
  4. the number of lines to go up or down is given by the method "getUnitsToScroll()"(from the mouse wheel event variable) as the mouse is scrolled up or down
  5. adding getUnitsToScroll() to FirstLineToDisplay gives you the new value that will be used to retrieve lines from the data table and send them to be displayed.
  6. you will need to check if the final value of FirstLineToDisplay is not below or above the data array size. If you try to retrieve lines that are not in the data table you will probably get an error message saying "array out of bounds exception"

Mini Project on Swing

Listening to the Headers

The listening to the headers functionality will be implemented using the Interface ActionListener.

- It will be implemented as an inner listener
  - check the lesson on inner listeners
- one ActionListener for each header button
- once the button is clicked it should be highlighted using a different color. All other buttons will then be reset to the standard color.



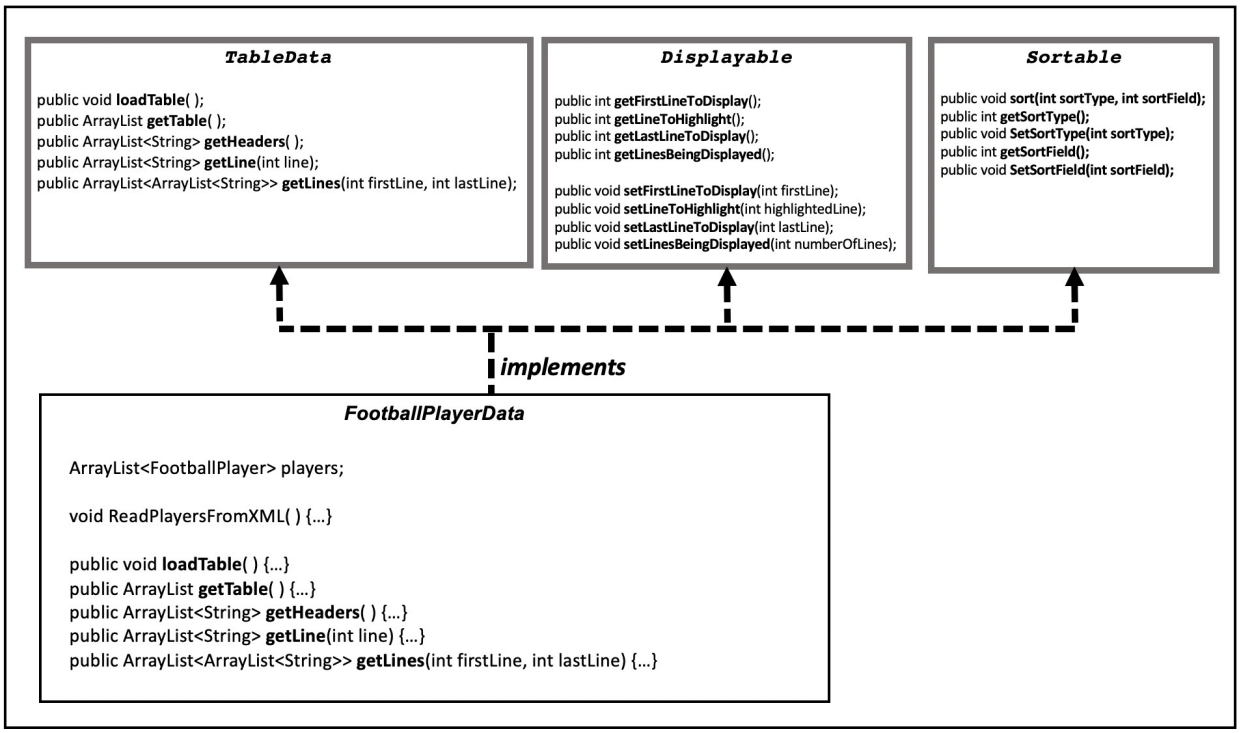
Mini Project on Swing

Part VII:

FootballPlayerData

FootballPlayerData will now implement a third interface, called sortable.

```
public class FootballPlayerData implements TableData, Displayable, Sortable
{
```



Functionality

Sorting the Data Table

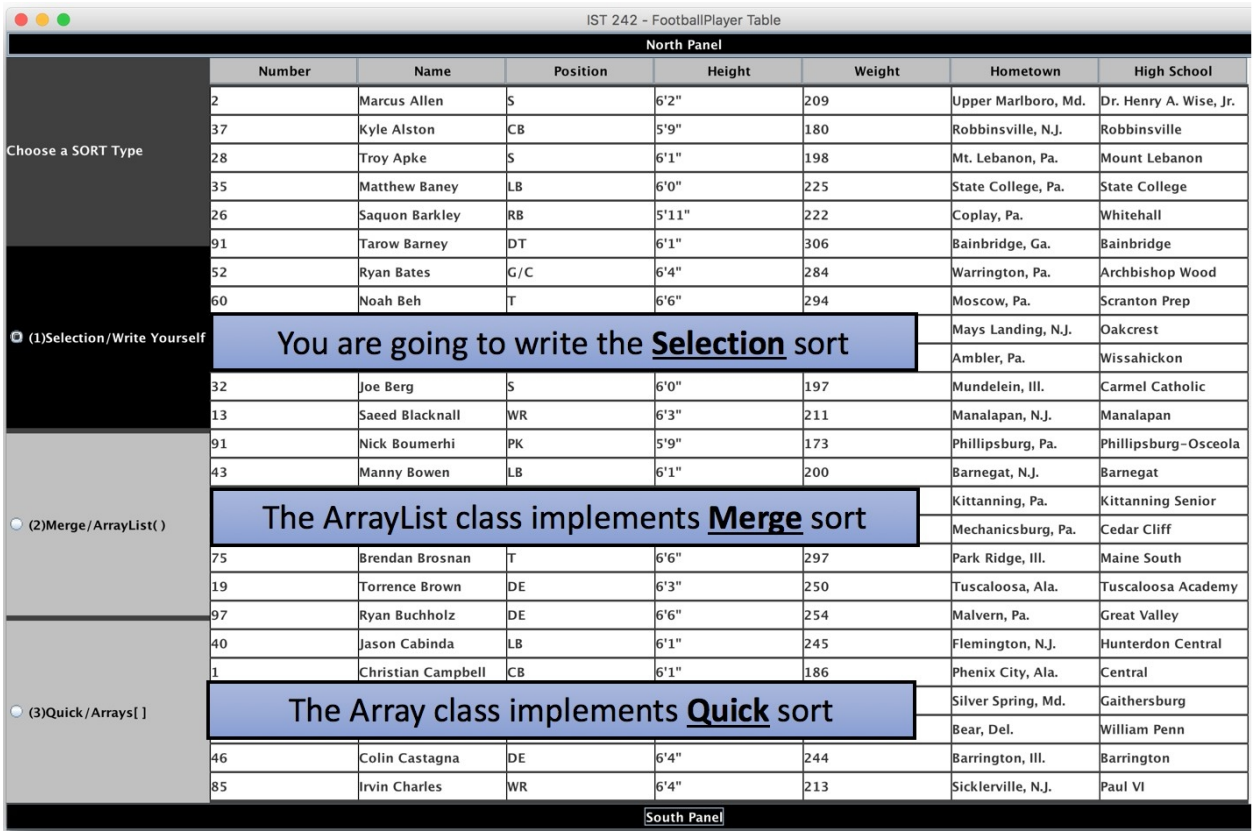
- The data table might be sorted by any of its attributes.
- The user will click on the header of a column and the table will be sorted by this attribute and displayed after being sorted.
- No need to sort by the inverse order if the header is clicked twice. Anyway, this is a nice feature to be implemented if you want but it is not required.

Choosing the sort algorithm

- The user will also be able to choose which sort algorithm will be used.
- Merge sort, which is the algorithm implement by Collections.sort(ARRAYLIST) (see lesson on sort)
- Quick sort, which is the algorithm implement by Arrays.sort(ARRAY) (see lesson on sort)

Mini Project on Swing

- Selection sort, which you will write and implement yourself based on the textbook (see textbook).

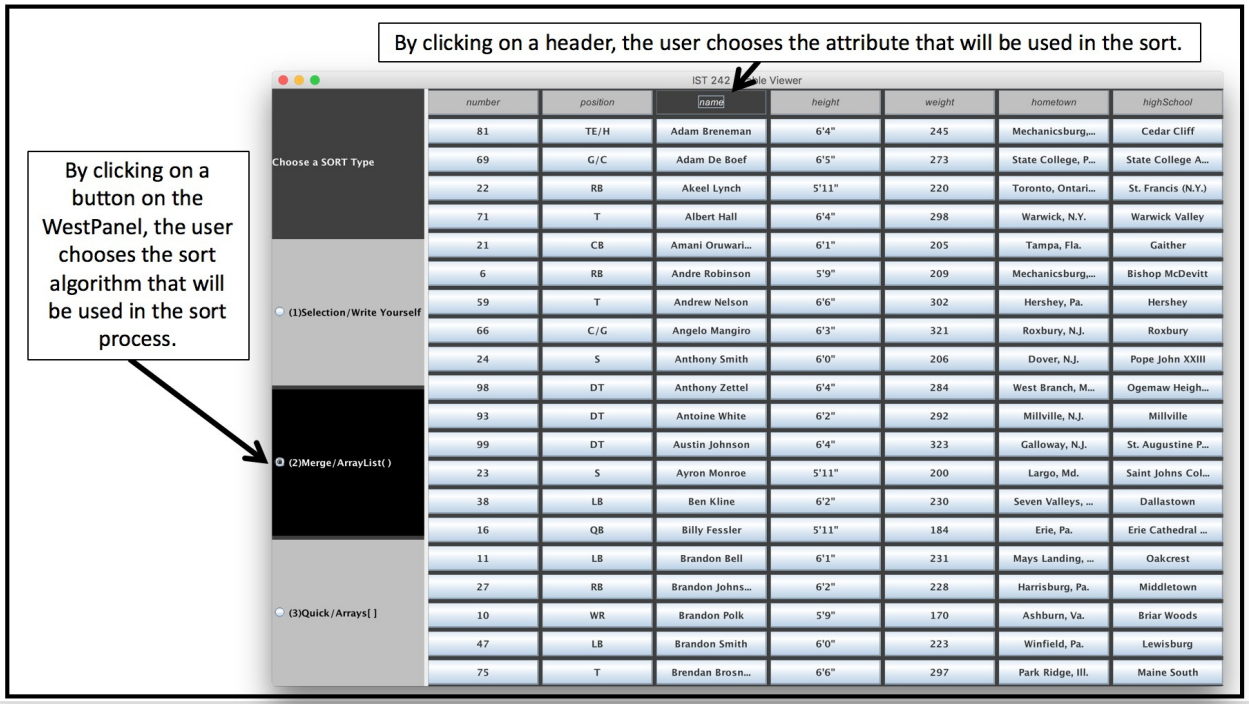


Number	Name	Position	Height	Weight	Hometown	High School
2	Marcus Allen	S	6'2"	209	Upper Marlboro, Md.	Dr. Henry A. Wise, Jr.
37	Kyle Alston	CB	5'9"	180	Robbinsville, N.J.	Robbinsville
28	Troy Apke	S	6'1"	198	Mt. Lebanon, Pa.	Mount Lebanon
35	Matthew Baney	LB	6'0"	225	State College, Pa.	State College
26	Saquon Barkley	RB	5'11"	222	Coplay, Pa.	Whitehall
91	Tarow Barney	DT	6'1"	306	Bainbridge, Ga.	Bainbridge
52	Ryan Bates	G/C	6'4"	284	Warrington, Pa.	Archbishop Wood
60	Noah Beh	T	6'6"	294	Moscow, Pa.	Scranton Prep
					Mays Landing, N.J.	Oakcrest
					Ambler, Pa.	Wissahickon
32	Joe Berg	S	6'0"	197	Mundelein, Ill.	Carmel Catholic
13	Saeed Blacknall	WR	6'3"	211	Manalapan, N.J.	Manalapan
91	Nick Boumerhi	PK	5'9"	173	Phillipsburg, Pa.	Phillipsburg-Osceola
43	Manny Bowen	LB	6'1"	200	Barnegat, N.J.	Barnegat
					Kittanning, Pa.	Kittanning Senior
					Mechanicsburg, Pa.	Cedar Cliff
75	Brendan Brosnan	T	6'6"	297	Park Ridge, Ill.	Maine South
19	Torrence Brown	DE	6'3"	250	Tuscaloosa, Ala.	Tuscaloosa Academy
97	Ryan Buchholz	DE	6'6"	254	Malvern, Pa.	Great Valley
40	Jason Cabinda	LB	6'1"	245	Flemington, N.J.	Hunterdon Central
1	Christian Campbell	CB	6'1"	186	Phenix City, Ala.	Central
					Silver Spring, Md.	Gaithersburg
					Bear, Del.	William Penn
46	Colin Castagna	DE	6'4"	244	Barrington, Ill.	Barrington
85	Irvin Charles	WR	6'4"	213	Sicklerville, N.J.	Paul VI

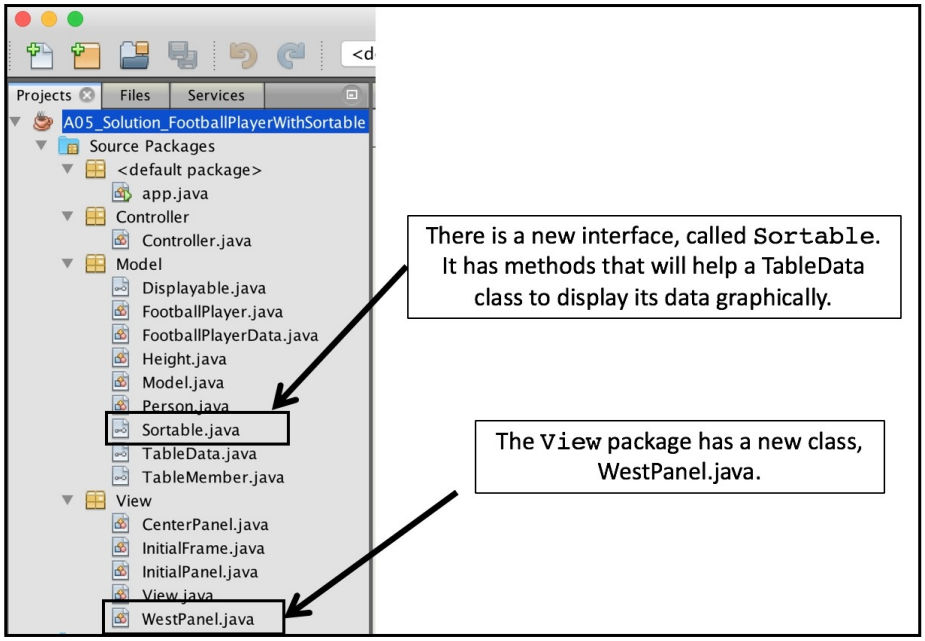
In order to use the default sorts (for ArrayLists and Arrays) you will have to use the interface Comparator (see lesson on sort).

The picture below show a selection based on radio buttons. You may choose whatever graphics mechanism you want, as long as the user are able to choose one type of sort.

Mini Project on Swing



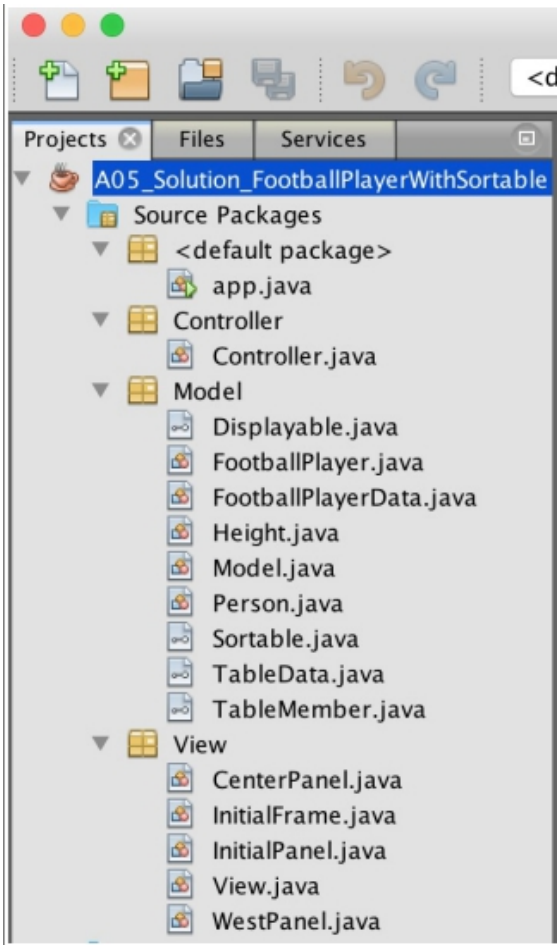
This new assignment brings in a new interface, Sortable, and a new class in View, WestPanel.



Mini Project on Swing

Create a Netbeans project with

- <default package>
  - App.java
- Model
  - Displayable.java
  - FootballPlayer.java
  - FootballPlayerData.java
  - Height.java
  - Model.java
  - Person.java
  - Sortable.java
  - TableData.java
  - TableMember.java
- Controller
  - Controller.java
- View
  - CenterPanel.java
  - InitialFrame.java
  - InitialPanel.java
  - View.java
  - WestPanel.java



The class

WestPanel

- it is the panel in charge of letting the user pick a sort algorithm
- it has a column of buttons or radio buttons (or any other choice mechanism)

The Interface

Sortable

These are the five methods that FootballPlayerData has to implement.

- public void sort(int sortType, int sortField);
  - this is the method that will execute the sort itself. It will choose the sort algorithm based on the input parameter sortType.
  - it will choose the field to the be sort key based on the input parameter sortField.
- public int getSortType();
  - returns an int number (1,2 or 3) to represent the user choice of algorithm updating the sortType attribute.
    - (1) Selection/Write Yourself
    - (2)Merge/ArrayList( )

- (3)Quick/Arrays[ ]

- public void SetSortType(int sortType);
  - defines a new value for sortType
- public int getSortField();
  - returns an int number (0 to n) to represent the user choice of the field to be used as the sort key.
- public void SetSortField(int sortField);
  - defines a new value for sortField

the code in Sortable

```
package  Model;

public  interface  Sortable
{
    public  void  sort(int  sortType,  int  sortField);
    public  int  getSortType();
    public  void  SetSortType(int  sortType);
    public  int  getSortField();
    public  void  SetSortField(int  sortField);
}
```

```
sortingOut()
{
    createCourses();
    Collections.shuffle(courseTable);
    System.out.println("SHUFFLED");
    System.out.println(courseTable.toString());
    System.out.println("====================================");
    //COMPARATOR//
    Comparator<course> sortCourseByName = new Comparator<course>()
    {
        @Override
        public int compare(course c1, course c2)
        {
            return c1.name.compareTo(c2.name);
        }
    };
    Comparator<course> sortCourseByNumber = new Comparator<course>()
    {
        @Override
        public int compare(course c1, course c2)
        {
            if (c1.number < (c2.number))
            {
                return -1;
            }
            if (c1.number == (c2.number))
            {
                return 0;
            }
            return 1;
        }
    };

    Collections.sort(courseTable,sortCourseByName);
    System.out.println("SORTED by Name");
    System.out.println(courseTable.toString());
    System.out.println("==============================
```

You can create comparators

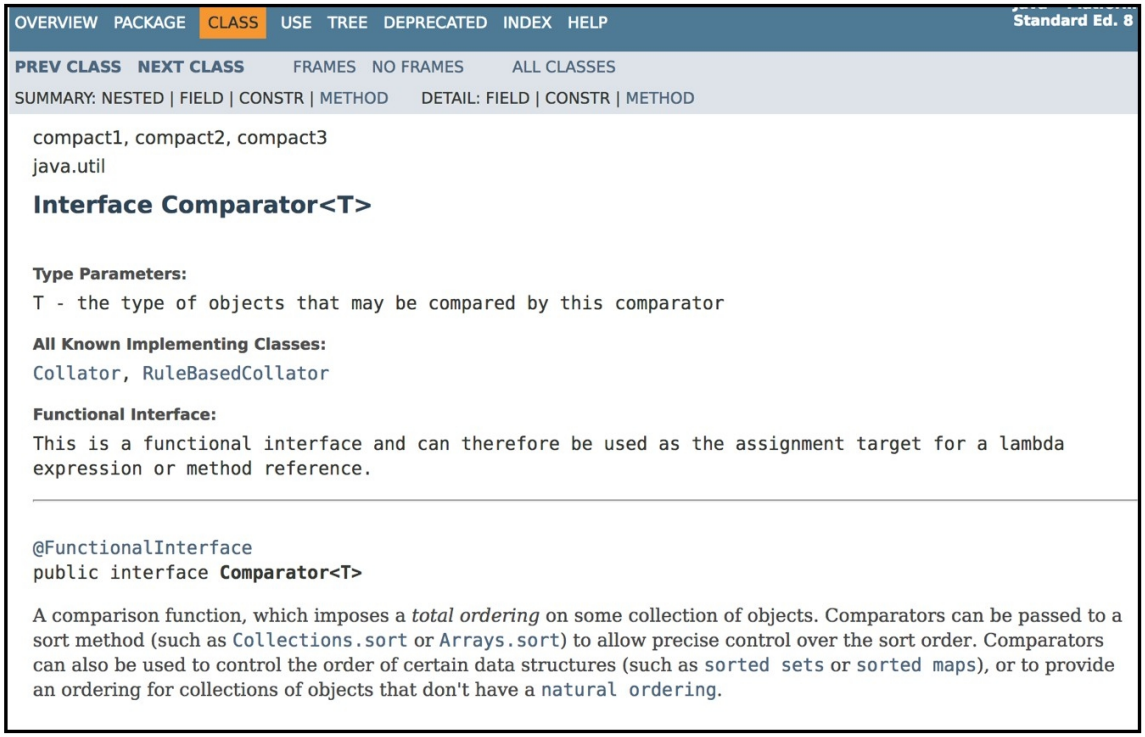
You choose which one to use when you call the default **sort** method by using a comparator as an input parameter

Mini Project on Swing

The Interface Comparator

See the Sort Lesson and the textbook for a detailed explanation for the Comparator interface.

Comparator is an important Java interface that is used to support sorting operations. Each field in the table will have its own comparator. The comparator interface has one abstract method, compare, that returns 0, 1 or -1 when comparing the two objects that it receives as input parameters.



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.util

Interface Comparator<T>

Type Parameters:

T - the type of objects that may be compared by this comparator

All Known Implementing Classes:

Collator, RuleBasedCollator

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public interface Comparator<T>
```

A comparison function, which imposes a *total ordering* on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering.