Mobile Application Design
Building a TestPlatform Project

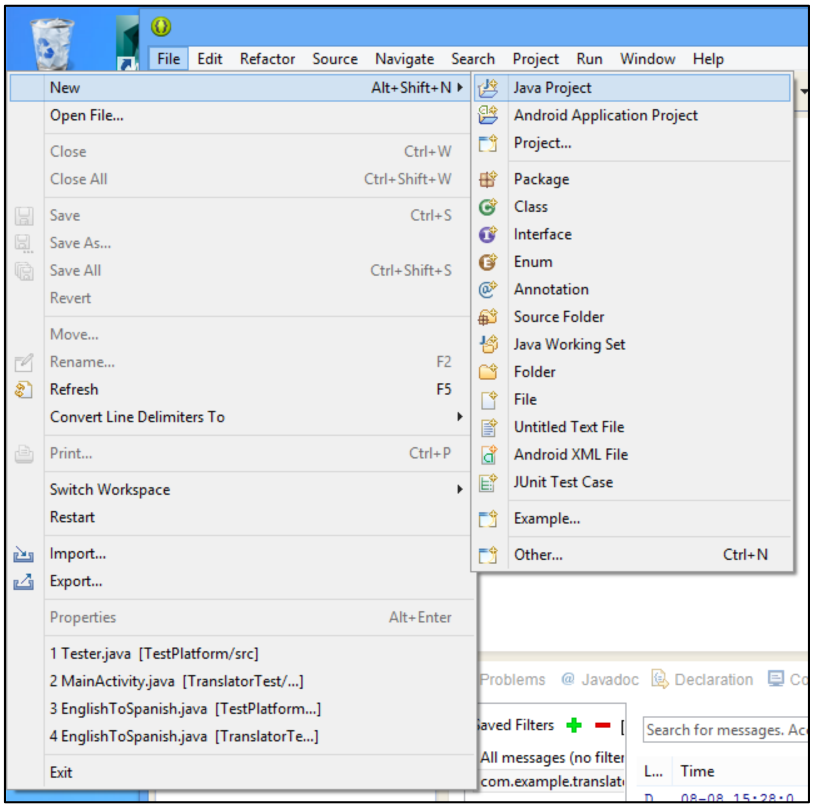
Description:

During the App development process it will be helpful to have platform that allows you to test the functionality of classes and code without having to completely load it into an Android emulator or device. For example, if you were writing a translation app, you would want to test the process of translating strings of words before you worked on user interface, text to speech, and layout objects in Android.

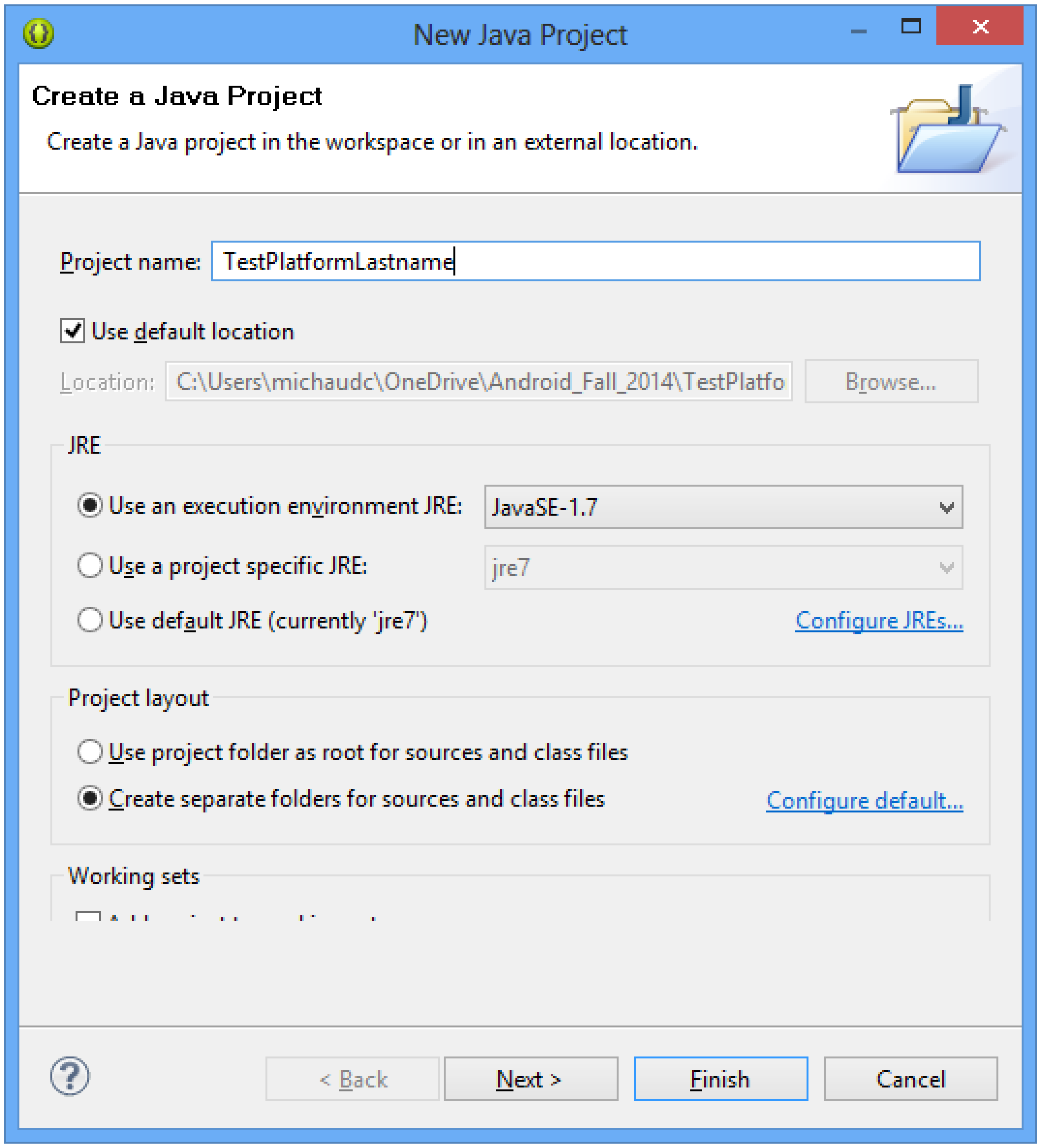
This project will walk you through the creation of a TestPlatform project in Java only. Then we will create some sample classes and test the inputs and outputs. Finally, we will begin to write an English to Spanish translator class we will use in a later App design project.

Process:

1. Start Eclipse and select "File – New – Java Project" (Note this is NOT an Android project)



2. Name the project "TestPlatformLastname". Click "Finish"


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: TestPlatformLastname

[✓] Use default location

Location: C:\Users\michaudc\OneDrive\Android_Fall_2014\TestPlatfo

JRE

● Use an execution environment JRE: JavaSE-1.7

○ Use a project specific JRE: jre7

○ Use default JRE (currently 'jre7')

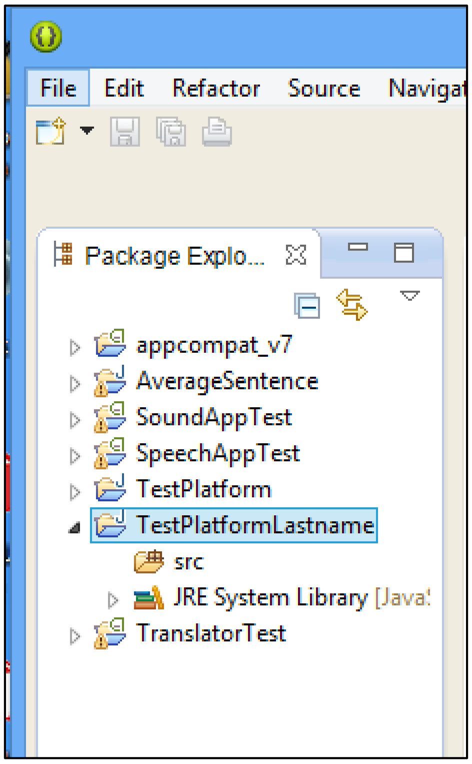
Project layout

○ Use project folder as root for sources and class files

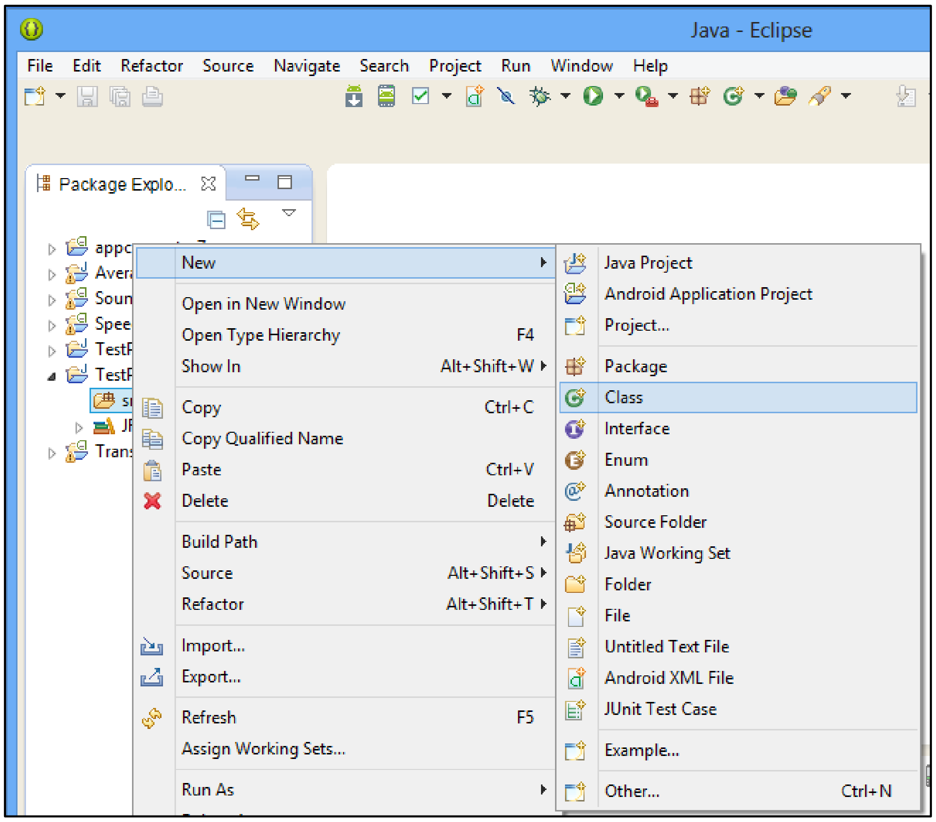
● Create separate folders for sources and class files

Working sets

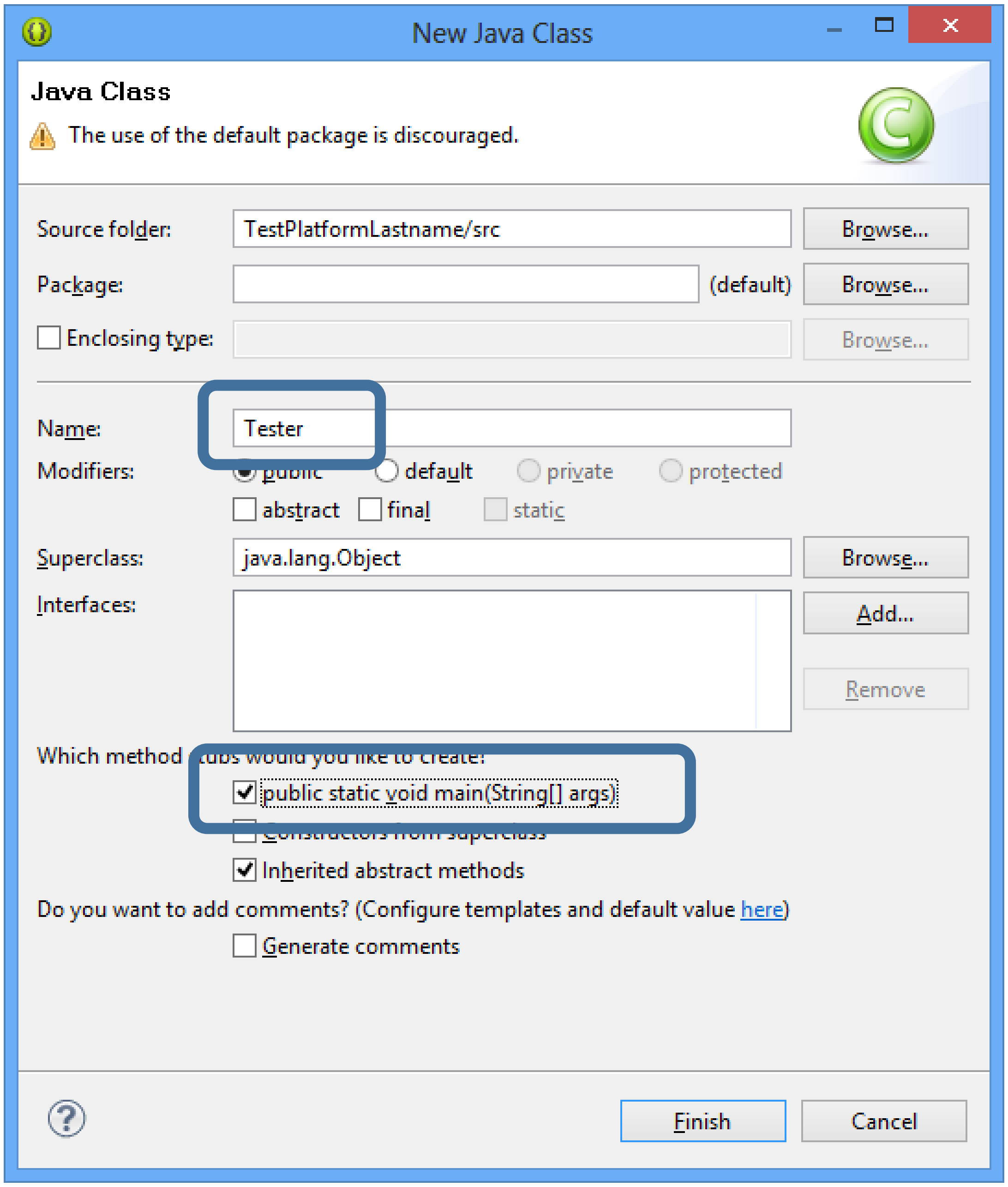
3. Expand the TestPlatformLastname project in the Package Explorer by clicking on the triangle.



4. We will now create a new class called ‘Tester’. This class acts as the location where we will test code and classes. Right click on the ‘src’ folder and select “New – class”



5. Name the class ‘Tester’. Check the box that adds a public static void method. Click ‘Finish’.


New Java Class

Java Class

The use of the default package is discouraged.

Source folder: TestPlatformLastname/src

Package: (default)

Enclosing type: [ ]

Name: Tester

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

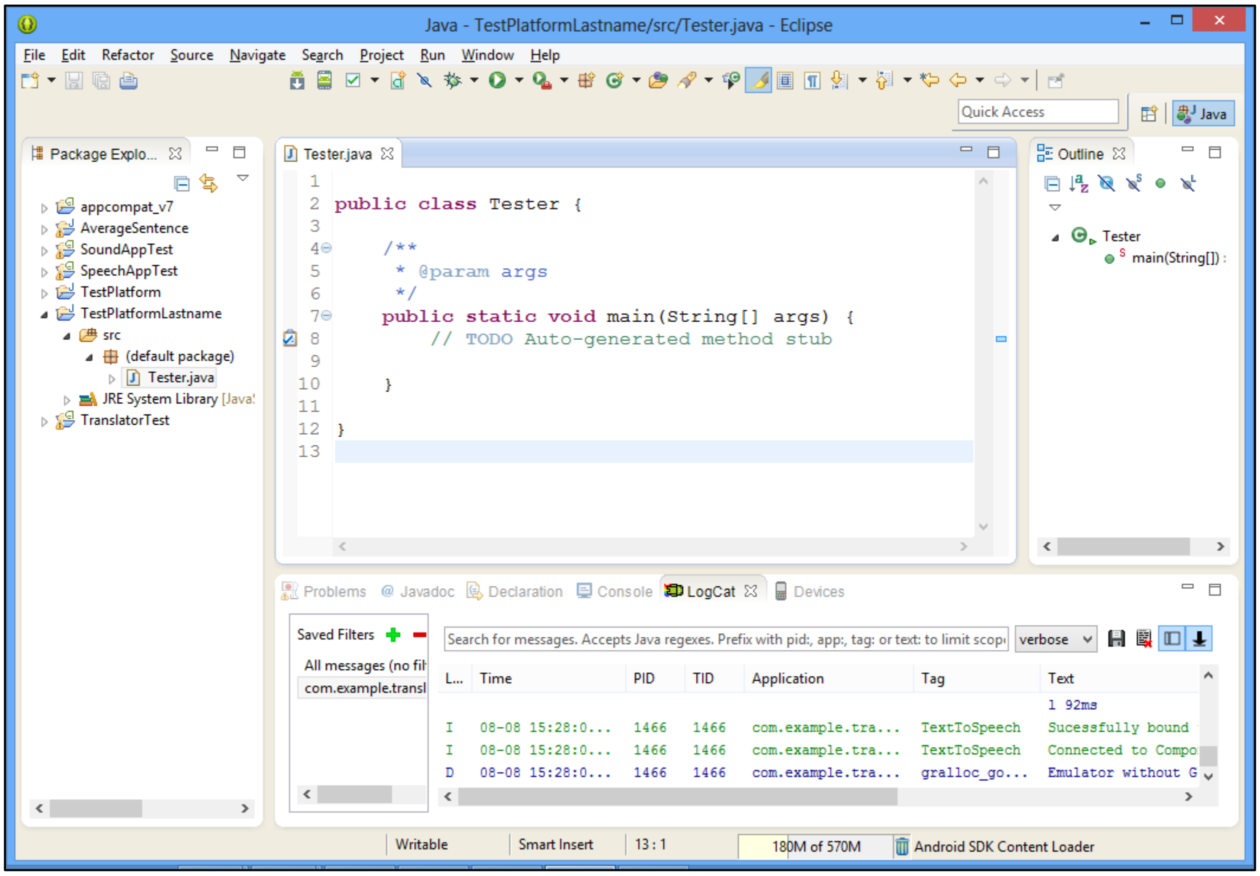
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

6. The ‘Tester.java’ class will open and display several comments and a ‘main’ method



```
public class Tester {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

```
L...  Time             PID    TID    Application          Tag             Text
                                                                           1 92ms
I     08-08 15:28:0...  1466   1466   com.example.tra...   TextToSpeech    Sucessfully bound
I     08-08 15:28:0...  1466   1466   com.example.tra...   TextToSpeech    Connected to Compo
D     08-08 15:28:0...  1466   1466   com.example.tra...   gralloc_go...   Emulator without G
```

7. The ‘public static void main’ is the main method of this Java class. Any code typed into this method will be run when the program starts. Android Java applications to not have a ‘main’ method as the Activity or ActionBarActivity classes extend the Android libraries within the structure of an application. Here in the ‘Tester’ class we place the commands we want to run in the ‘main’ method.

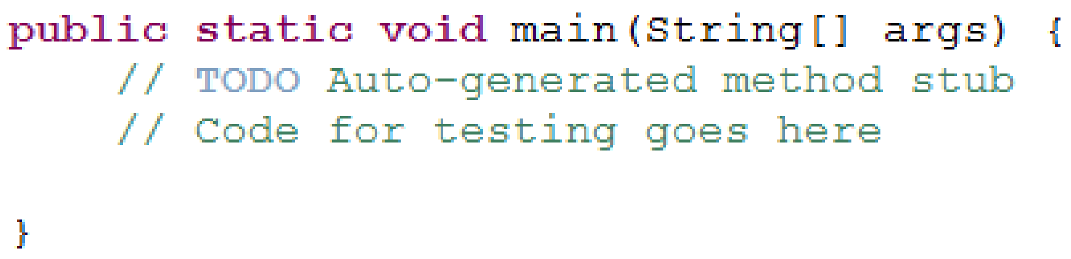
Java key word meanings:

public:
All objects in program can access

static:
Related to class type, not instance

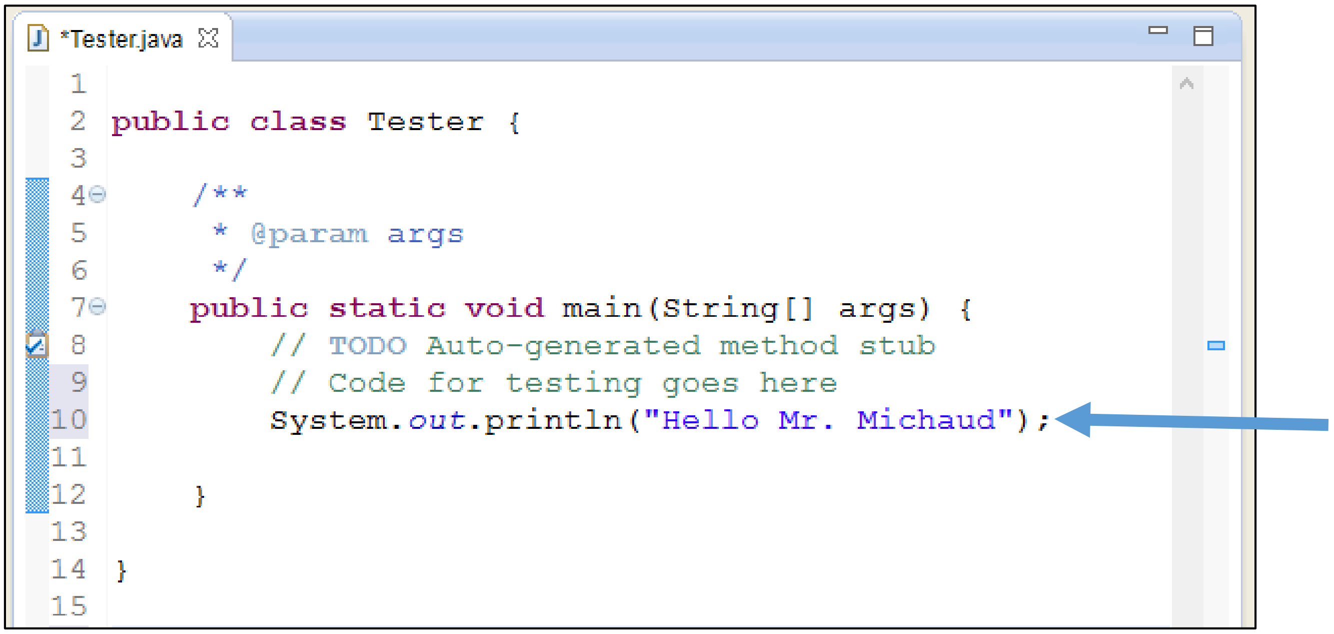
void:
Return type. Does not give data back to the program

main:
Name of method. Will run first when program executes

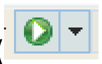


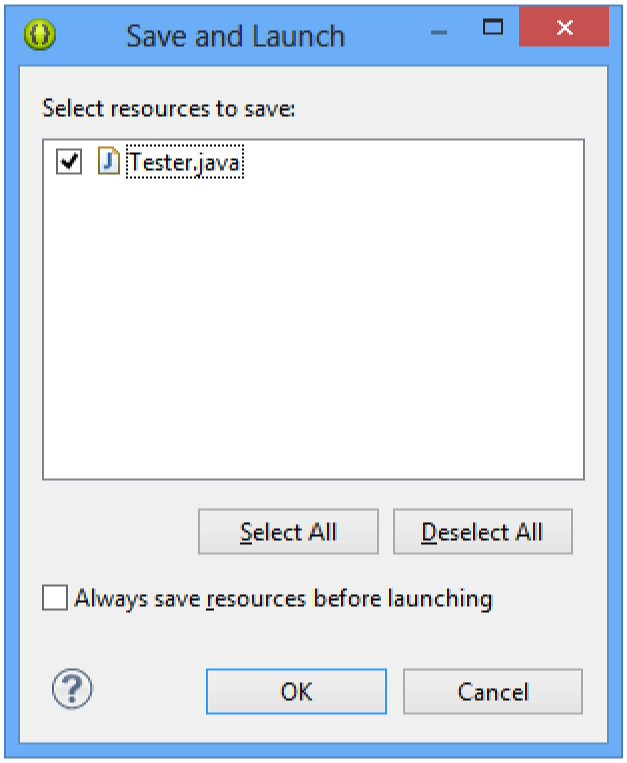
```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    // Code for testing goes here

}
```

8. Let us try a print text command. In Java, when we want to print data to the console (text output in IDE or computer), we use the command ‘System.out.println()’. Type the following inside the main method.



```

public class Tester {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        System.out.println("Hello Mr. Michaud");

    }

}

```

9. To run the program, click the run icon on the toolbar. (). A window should appear asking to save. Click “OK”.



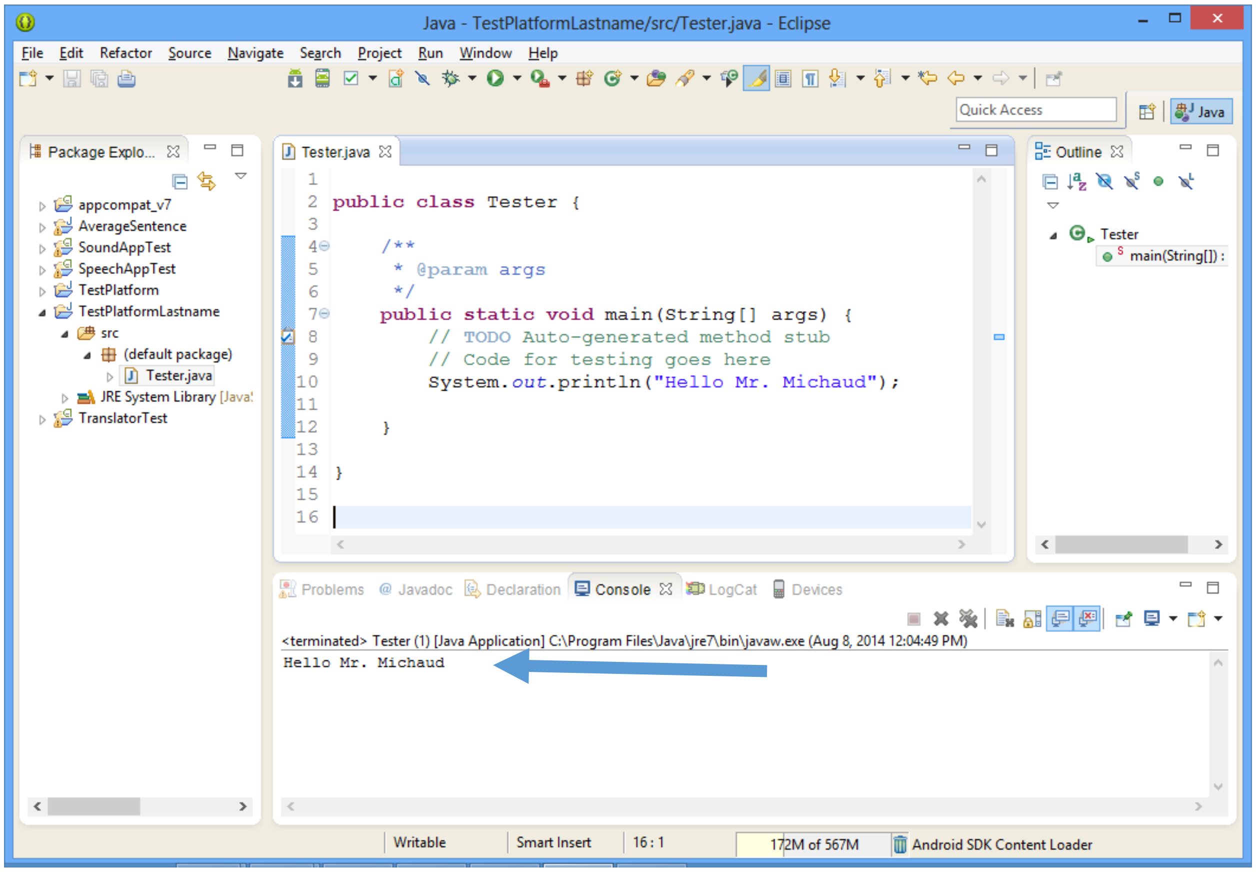
Save and Launch

Select resources to save:

☑ Tester.java

☐ Always save resources before launching

10. In the Console view – you should see the text we told the computer to print:

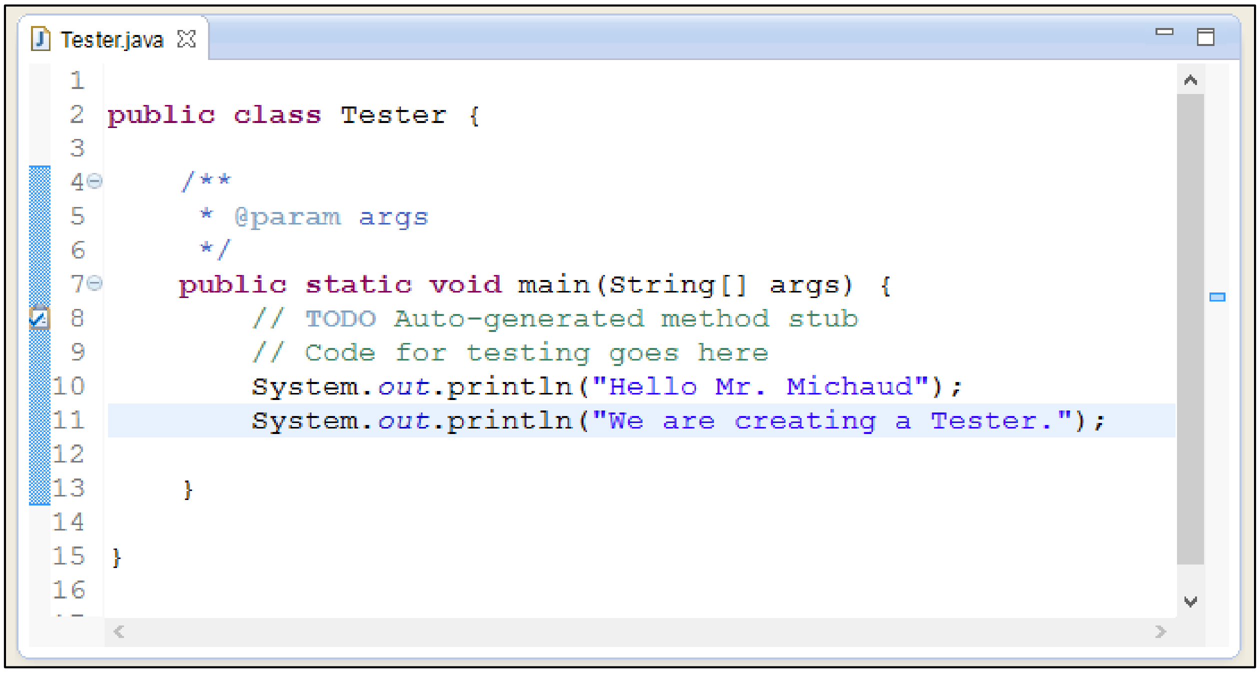


Java - TestPlatformLastname/src/Tester.java - Eclipse

```java
public class Tester {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        System.out.println("Hello Mr. Michaud");

    }

}

```

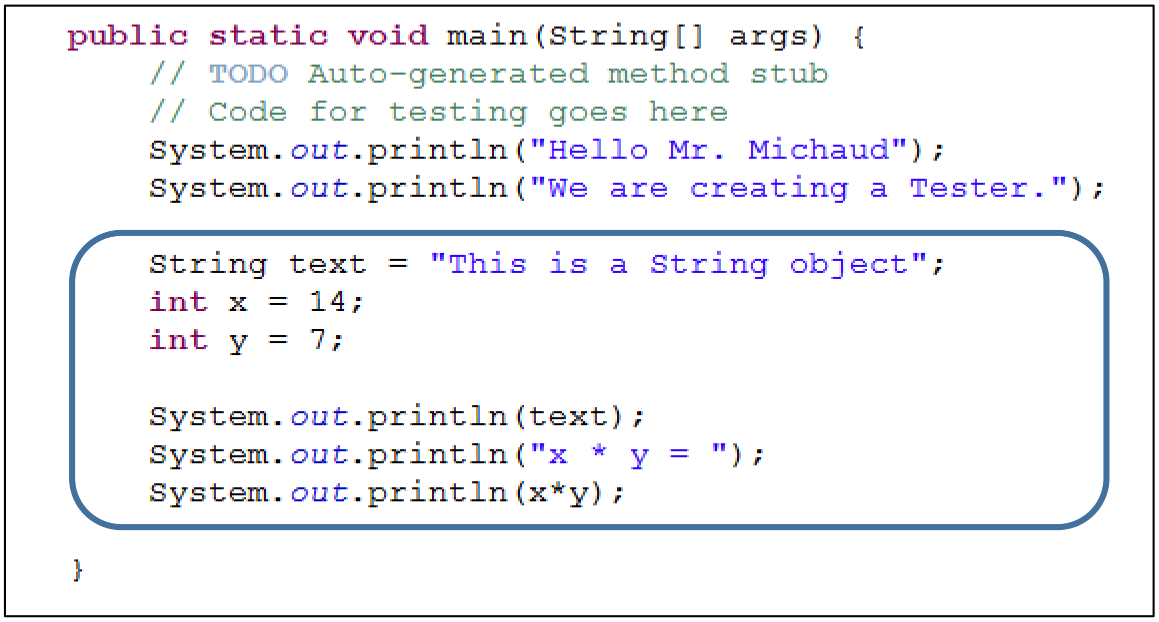
```
<terminated> Tester (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Aug 8, 2014 12:04:49 PM)
Hello Mr. Michaud
```

11. Try another println command:



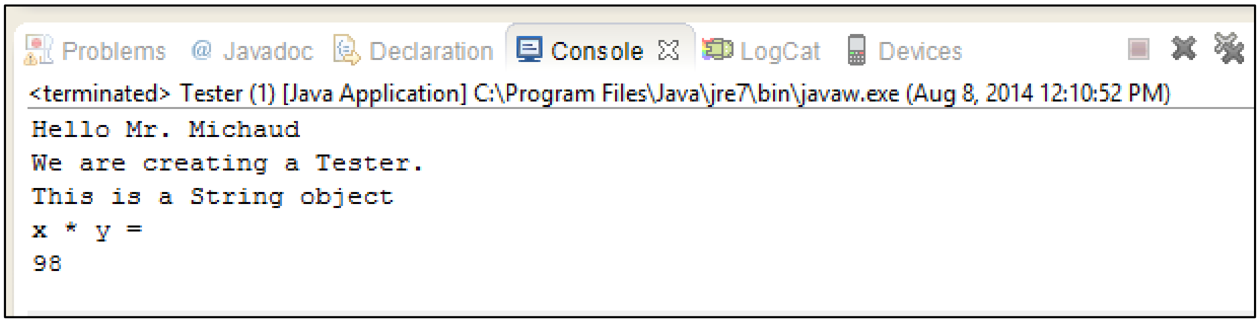
```java
public class Tester {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        System.out.println("Hello Mr. Michaud");
        System.out.println("We are creating a Tester.");

    }

}

```

12. We can also experiment with variables and datatypes. Add the following and run:



```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    // Code for testing goes here
    System.out.println("Hello Mr. Michaud");
    System.out.println("We are creating a Tester.");

    String text = "This is a String object";
    int x = 14;
    int y = 7;

    System.out.println(text);
    System.out.println("x * y = ");
    System.out.println(x*y);

}
```

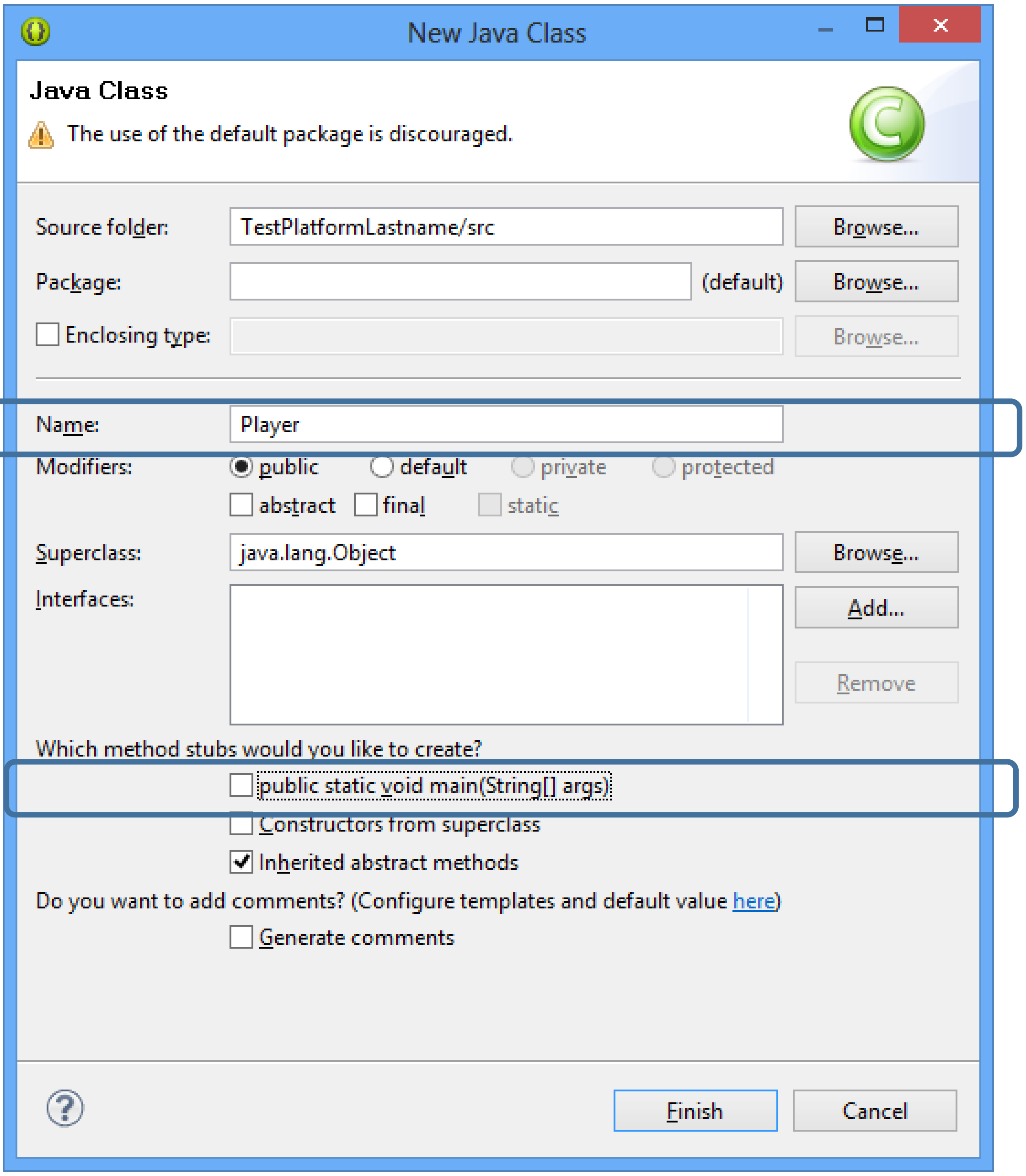
13. The result in the console should look like:



```
<terminated> Tester (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Aug 8, 2014 12:10:52 PM)
Hello Mr. Michaud
We are creating a Tester.
This is a String object
x * y =
98
```

14. We usually do not place all of our programming inside the main method. Instead, in Java we create classes that represent objects, their values, and their abilities (functions or methods). We are now going to create a 'Player' class that will hold a player's name and position.

Right click on the 'default package' icon and select 'New – Class'. Name the class 'Player'. Make sure the 'public static void main' is NOT checked.



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: TestPlatformLastname/src

Package: (default)

Enclosing type: ☐

Name: Player

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

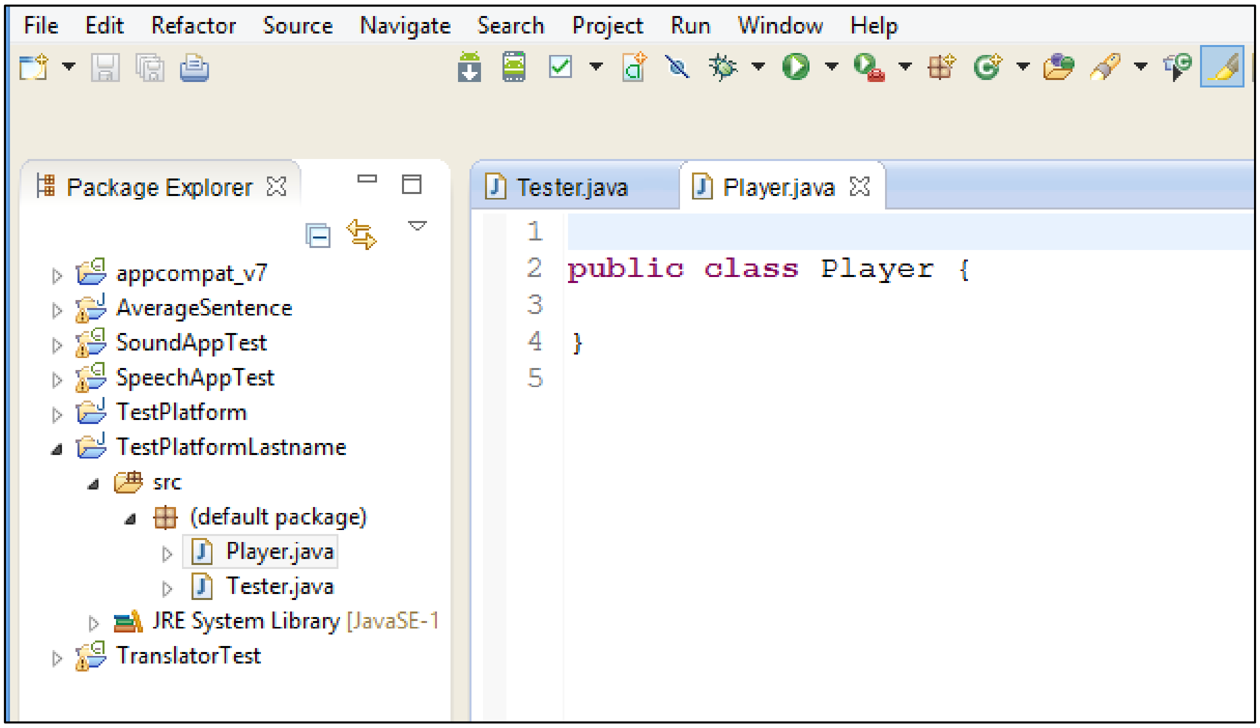
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

15. We now have two classes – Player and Tester in the package. Click on ‘Player’.

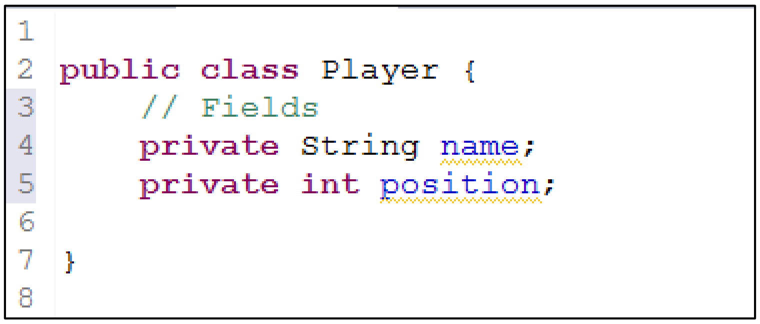


```java

public class Player {

}

```

16. We need to create the following design for a 'Player':

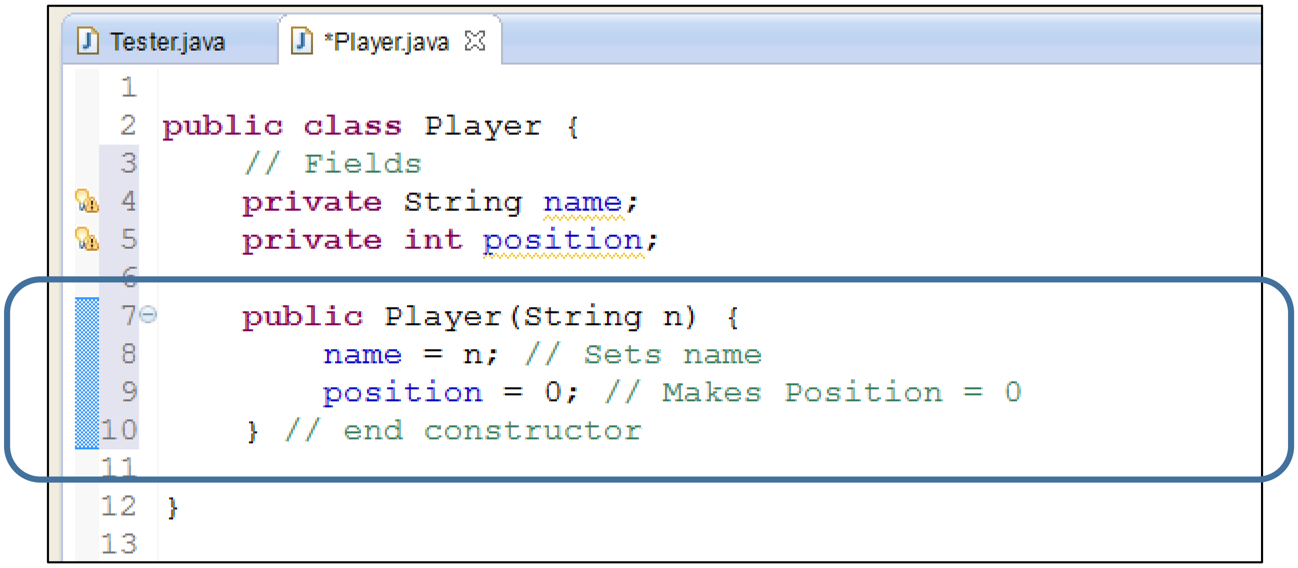
Player
Fields: Private String name Private integer position
Constructor: public Player(String name)
Methods: Public String getName Public int getPosition Public void setPosition(int p)

First, we will write the fields:



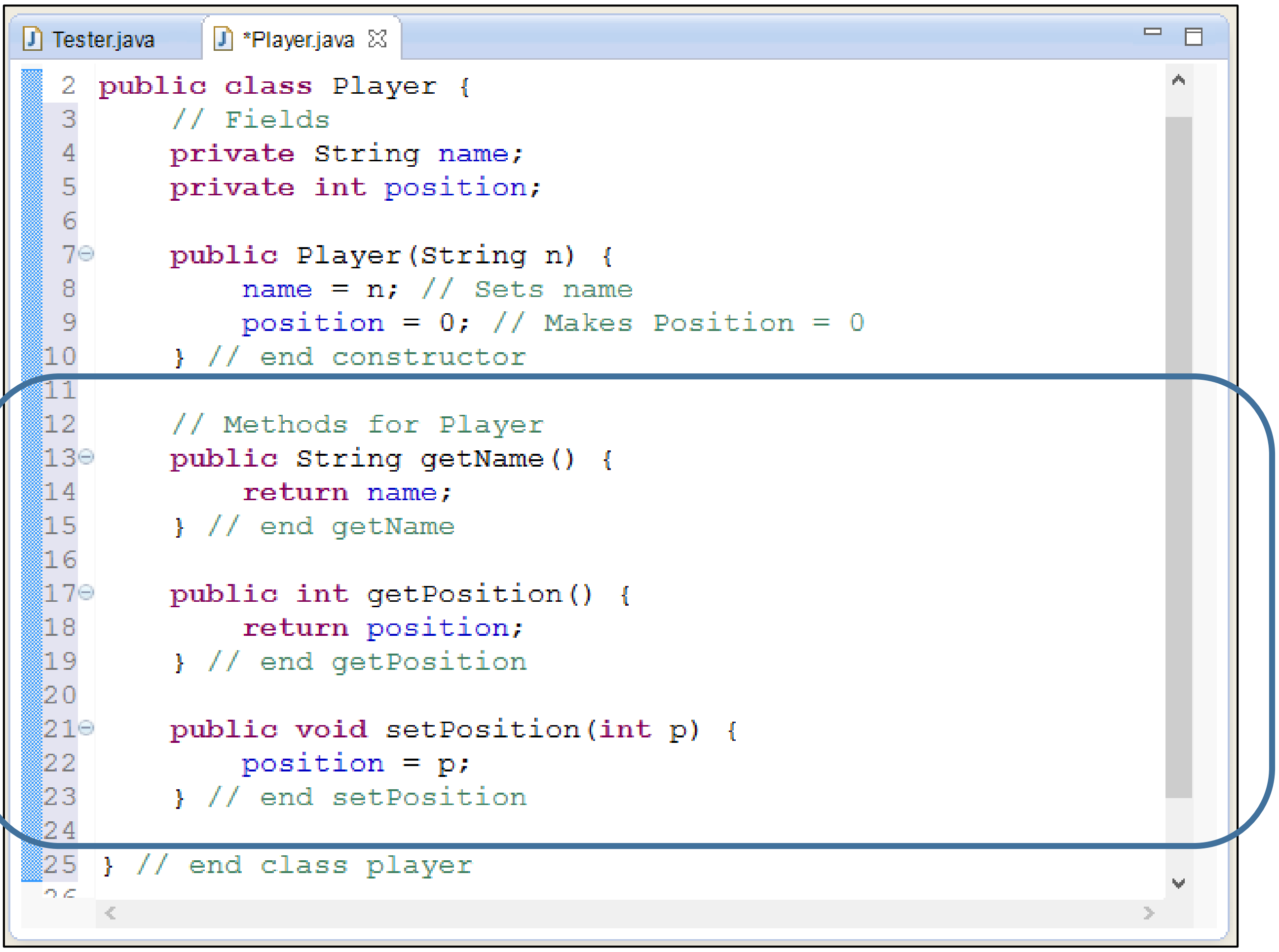
```java

public class Player {
    // Fields
    private String name;
    private int position;

}

```

17. The Constructor will build an instance of player and initialize the fields:



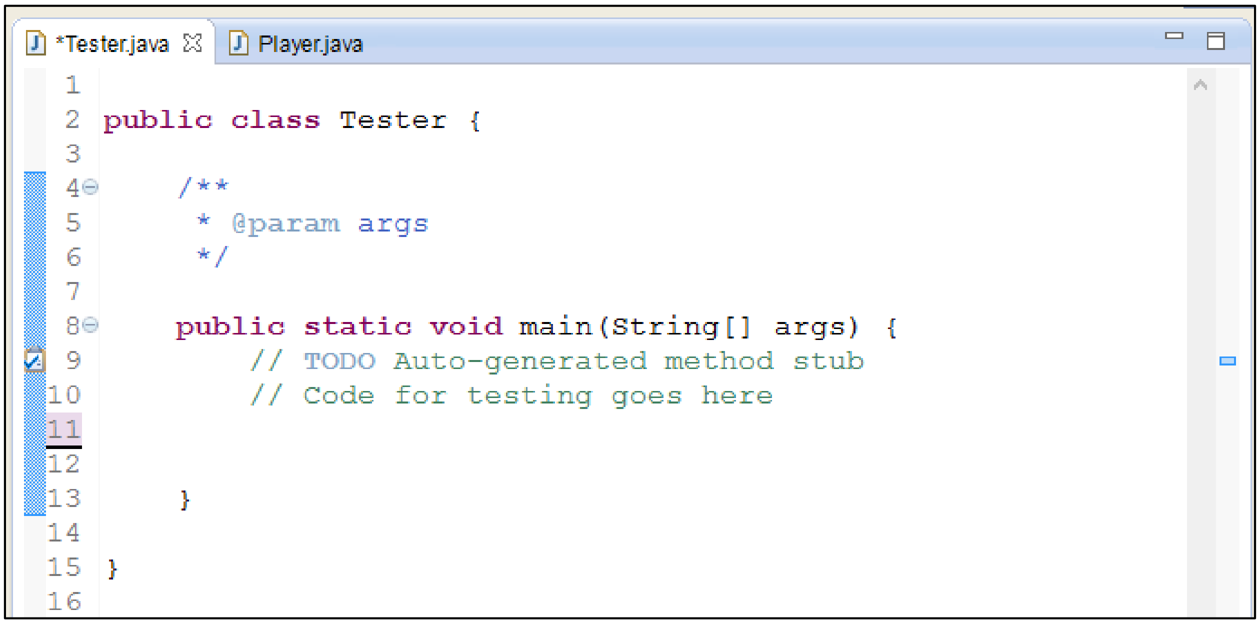
```java

public class Player {
    // Fields
    private String name;
    private int position;

    public Player(String n) {
        name = n; // Sets name
        position = 0; // Makes Position = 0
    } // end constructor

}

```

18. We will now define the methods:



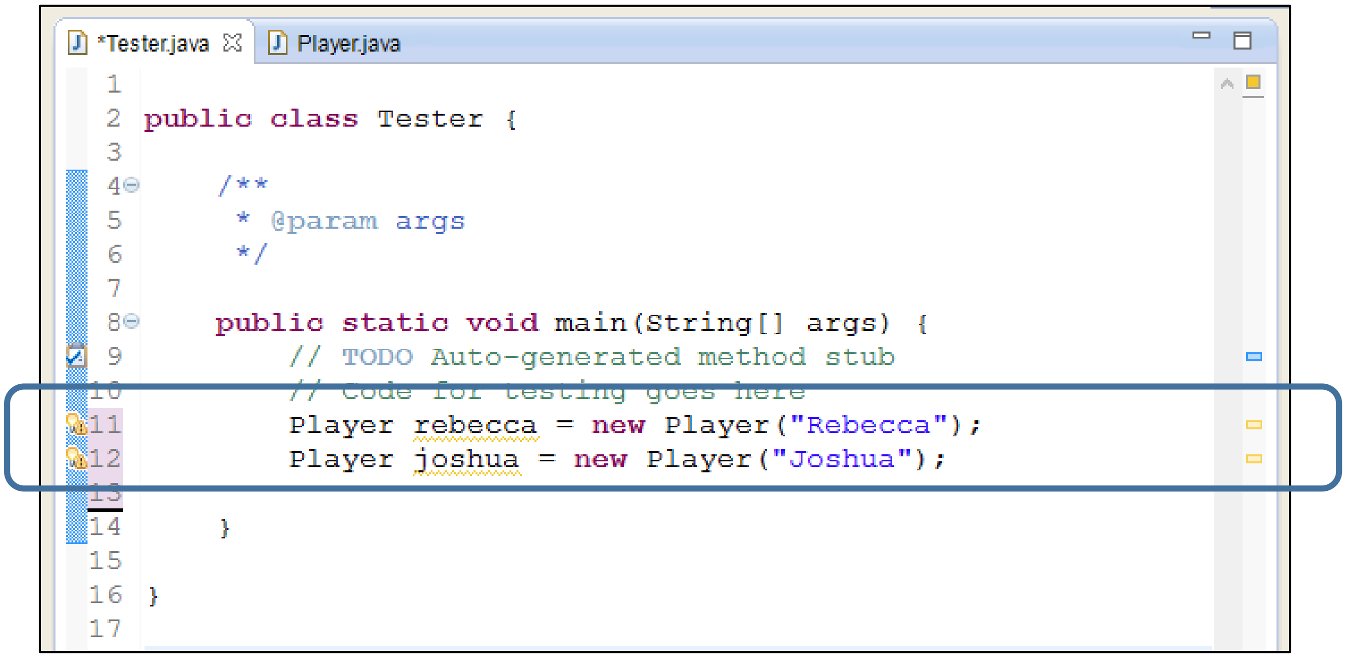
```java
public class Player {
    // Fields
    private String name;
    private int position;

    public Player(String n) {
        name = n; // Sets name
        position = 0; // Makes Position = 0
    } // end constructor

    // Methods for Player
    public String getName() {
        return name;
    } // end getName

    public int getPosition() {
        return position;
    } // end getPosition

    public void setPosition(int p) {
        position = p;
    } // end setPosition

} // end class player
```

19. Now that we have defined a Player, we will move back to the Tester class and run some tests with the Player class. Go to the Tester class and delete the code we wrote in the main method.



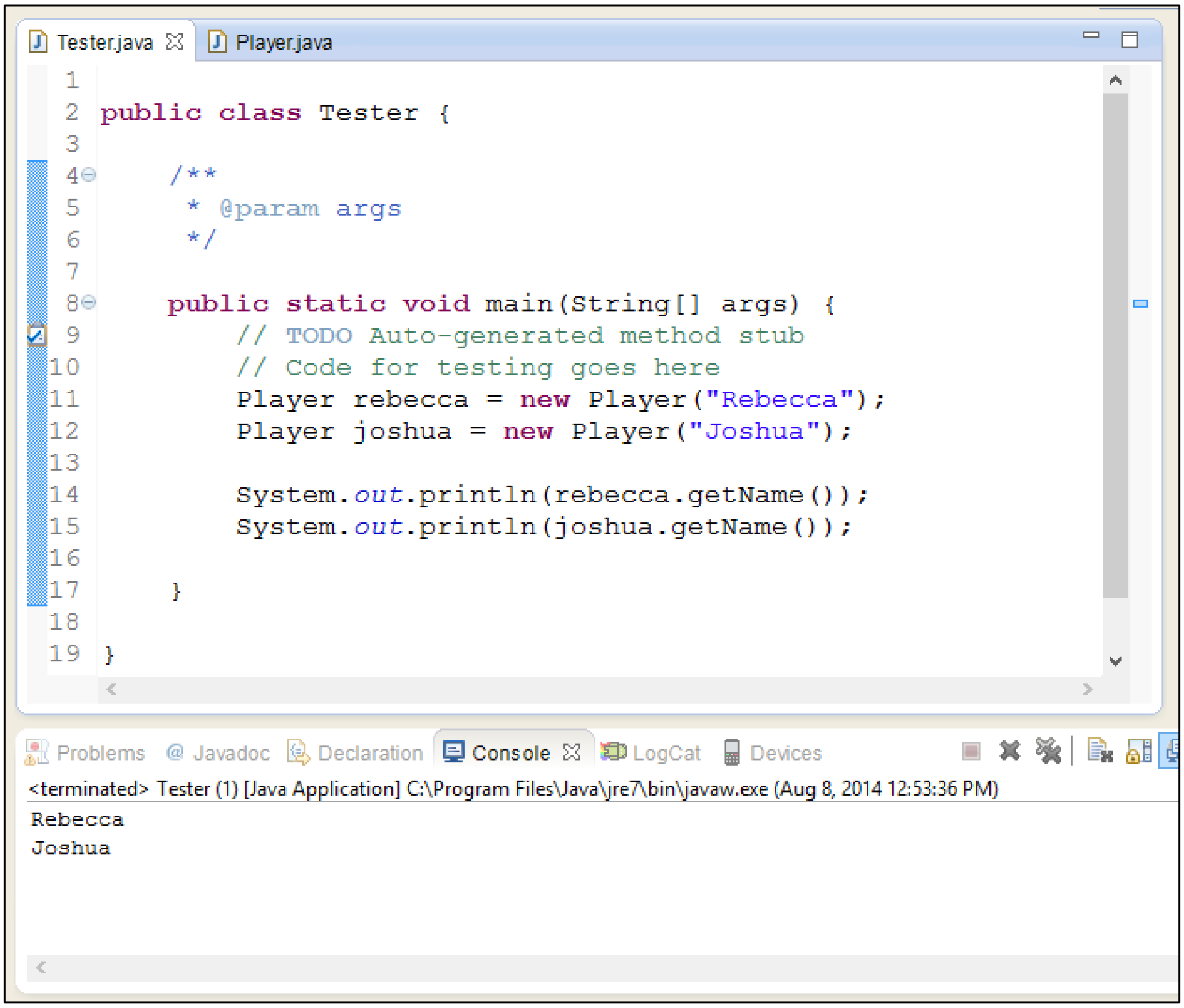
```

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here


    }

}

```

20. Now create two Player instances:



```

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        Player rebecca = new Player("Rebecca");
        Player joshua = new Player("Joshua");

    }

}

```

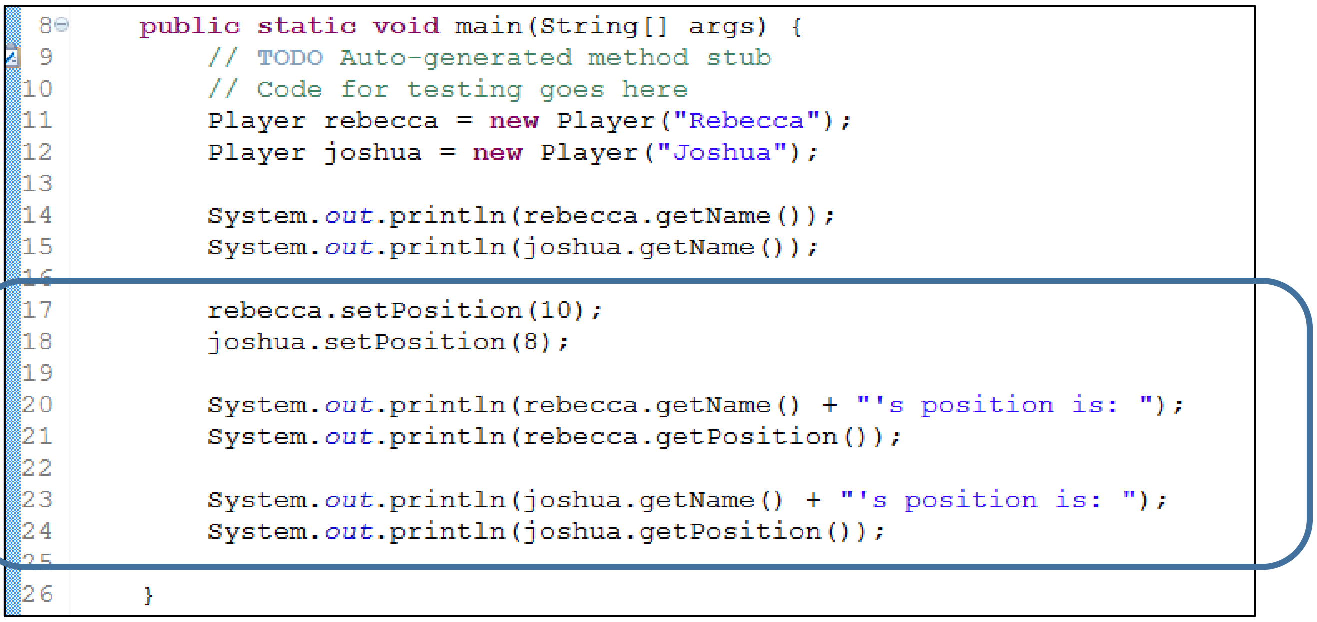
21. Our players know how to return their names. So, let us have ‘rebecca’ and ‘joshua’ print their names.



```java

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        Player rebecca = new Player("Rebecca");
        Player joshua = new Player("Joshua");

        System.out.println(rebecca.getName());
        System.out.println(joshua.getName());

    }

}
```

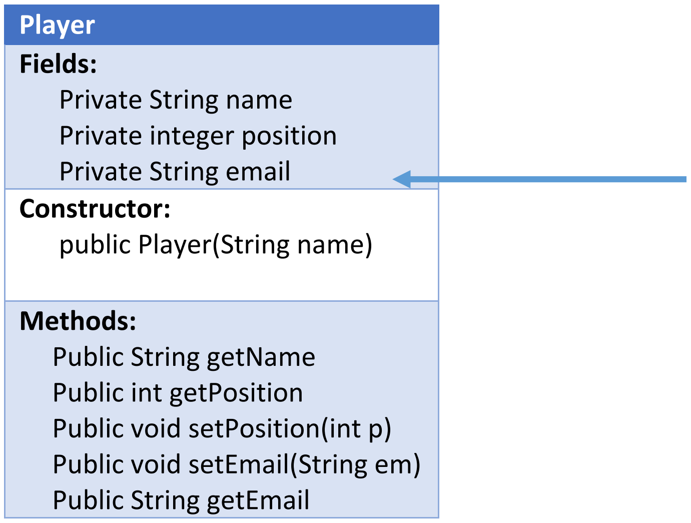
```
<terminated> Tester (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Aug 8, 2014 12:53:36 PM)
Rebecca
Joshua
```

22. We can also set and print the positions of ‘rebecca’ and ‘joshua’:



```java
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        Player rebecca = new Player("Rebecca");
        Player joshua = new Player("Joshua");

        System.out.println(rebecca.getName());
        System.out.println(joshua.getName());

        rebecca.setPosition(10);
        joshua.setPosition(8);

        System.out.println(rebecca.getName() + "'s position is: ");
        System.out.println(rebecca.getPosition());

        System.out.println(joshua.getName() + "'s position is: ");
        System.out.println(joshua.getPosition());

    }
```

23. Let us say that we wanted to add an ‘email’ field to the Player class:



Player
Fields: Private String name Private integer position Private String email
Constructor: public Player(String name)
Methods: Public String getName Public int getPosition Public void setPosition(int p) Public void setEmail(String em) Public String getEmail

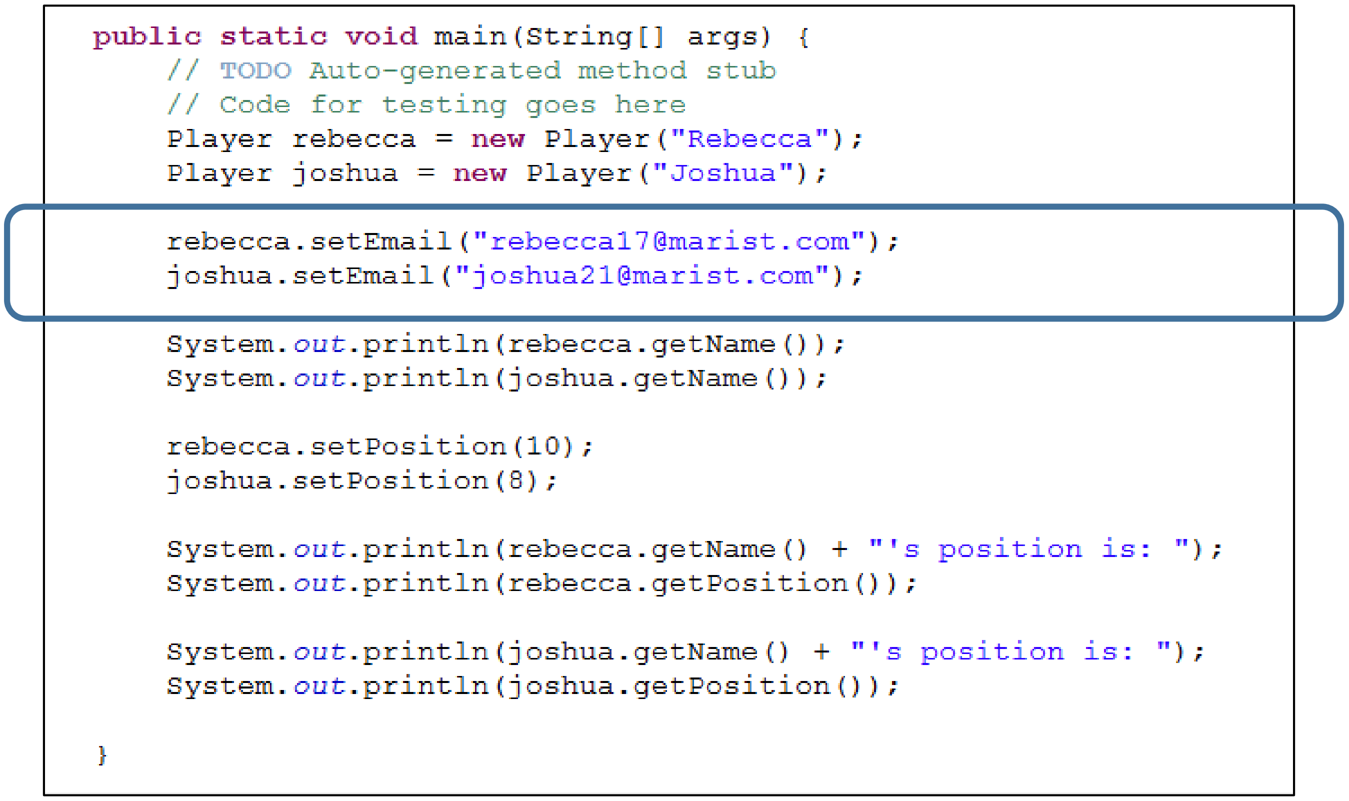
We will need to make three changes:

a. Add field ‘email’
b. Write method setEmail (modifier)
c. Write method getEmail (accessor)

Go to the Player class and add the email field and accessor / modifier methods.

```
public class Player {
     // Fields
     private String name;
     private int position;
     private String email;

     public Player(String n) {
          name = n; // Sets name
          position = 0; // Makes Position = 0
     } // end constructor

     // Methods for Player
     public String getName() {
          return name;
     } // end getName

     public int getPosition() {
          return position;
     } // end getPosition

     public void setPosition(int p) {
          position = p;
     } // end setPosition

     public void setEmail(String em) {
          email = em;
     }

     public String getEmail() {
          return email;
     }

} // end class player
```

24. Now we will modify the Tester class to add email addresses:



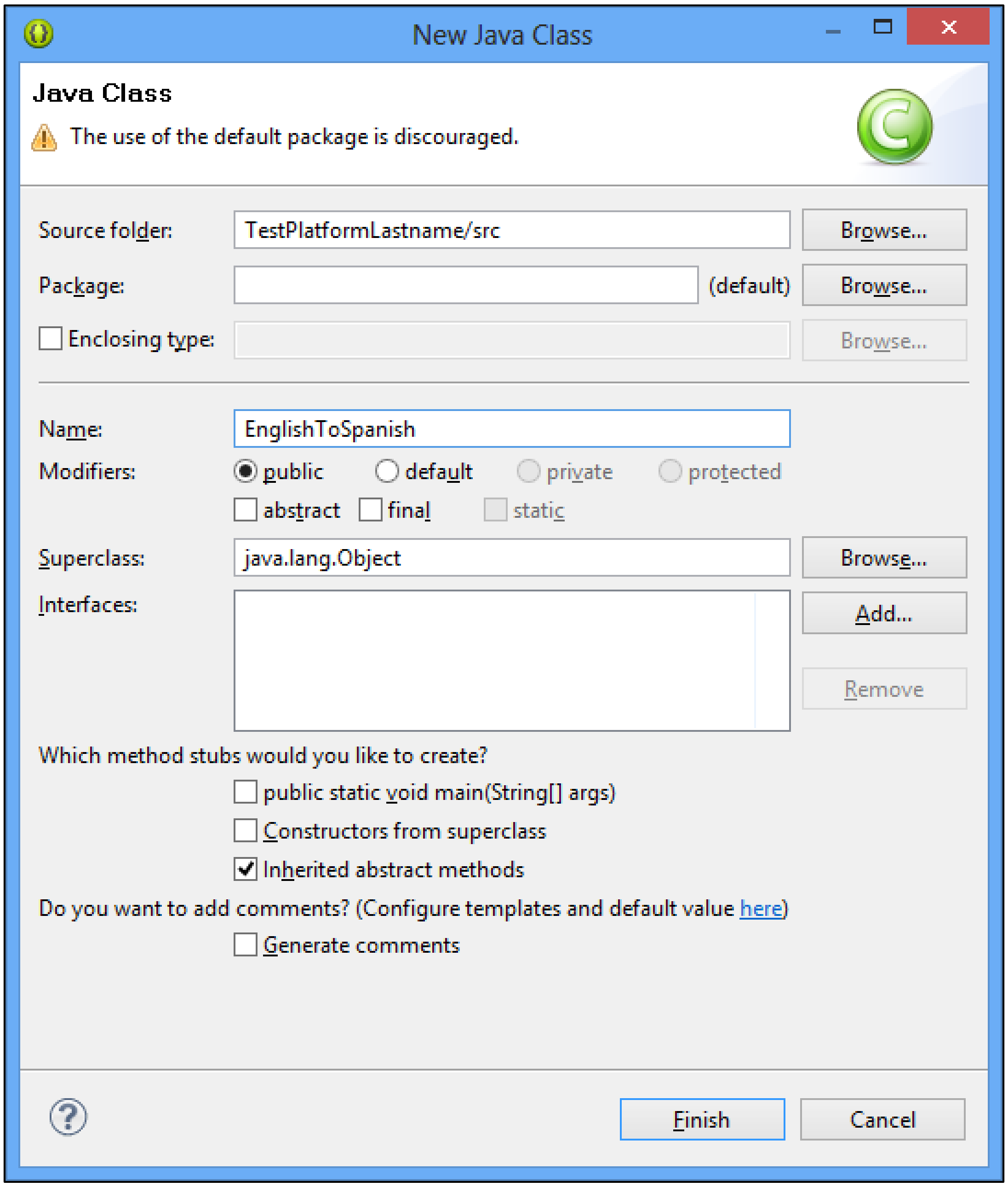
```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    // Code for testing goes here
    Player rebecca = new Player("Rebecca");
    Player joshua = new Player("Joshua");

    rebecca.setEmail("rebecca17@marist.com");
    joshua.setEmail("joshua21@marist.com");

    System.out.println(rebecca.getName());
    System.out.println(joshua.getName());

    rebecca.setPosition(10);
    joshua.setPosition(8);

    System.out.println(rebecca.getName() + "'s position is: ");
    System.out.println(rebecca.getPosition());

    System.out.println(joshua.getName() + "'s position is: ");
    System.out.println(joshua.getPosition());

}
```

25. Up to now we have practiced creating a class and storing data within fields of the class. Our next exercise will be to build a class that performs an action for the program, and returns a result. This class will be called 'EnglishToSpanish' and it will hold a List of words in English and Spanish and provide the ability to translate from English to Spanish using the word List.

EnglishToSpanish
Fields: Private List dictionary
Constructor: public EnglishToSpanish()
Methods: Public String translate(String s) -> Looks up word from List. Returns original word if word is not in dictionary list. Public void addEntry(String e, String s) -> User can add new English and Spanish words to list

26. The code for this class will be more involved and use the Java data structures of Lists and a for loop control structure. We will work with loops and Lists in detail later in the course, work now to get the code in and run tests. We will also use this EnglishToSpanish class later when we write an Translator App.

27. Right click on 'default package' and create a new class called 'EnglishToSpanish'. Make sure you DO NOT select 'public static void main' checkbox.


New Java Class

Java Class

The use of the default package is discouraged.

Source folder: TestPlatformLastname/src

Package: (default)

☐ Enclosing type:

Name: EnglishToSpanish

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

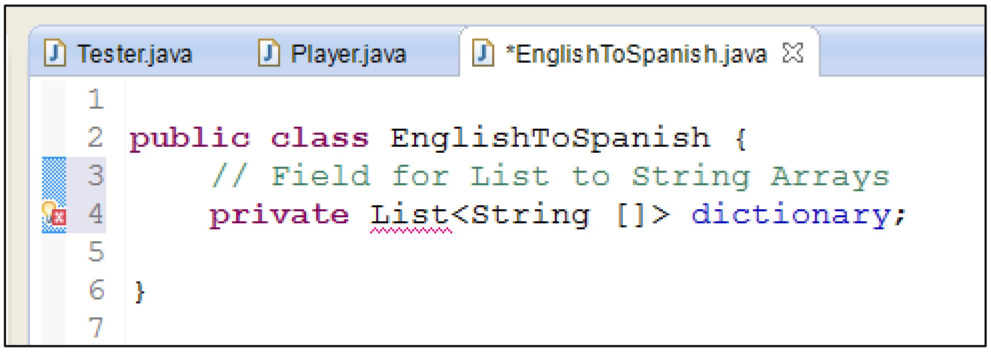
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

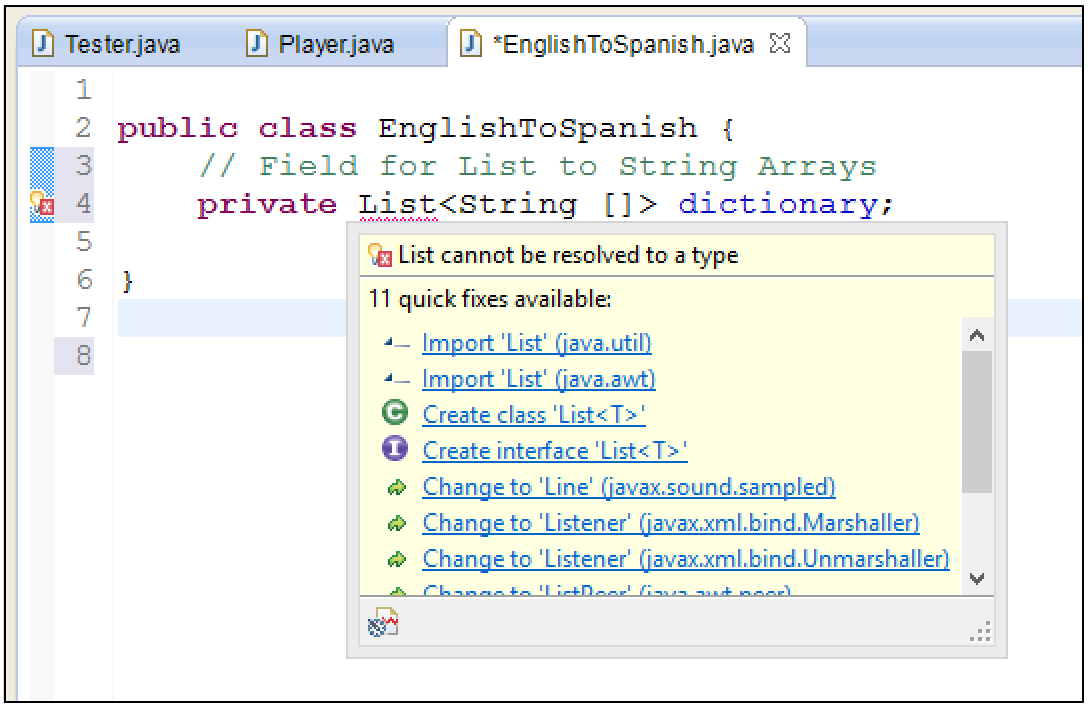
☐ Generate comments

28. Type the code to define the Field ‘dictionary’


Tester.java | Player.java | *EnglishToSpanish.java

```

public class EnglishToSpanish {
    // Field for List to String Arrays
    private List<String []> dictionary;

}

```

29. Note that the ‘List’ word is underlined in red. This is because a ‘List’ is a Java object we need to import. Hover over the word ‘List’ and select ‘Import ‘List’ (java.util).


Tester.java | Player.java | *EnglishToSpanish.java

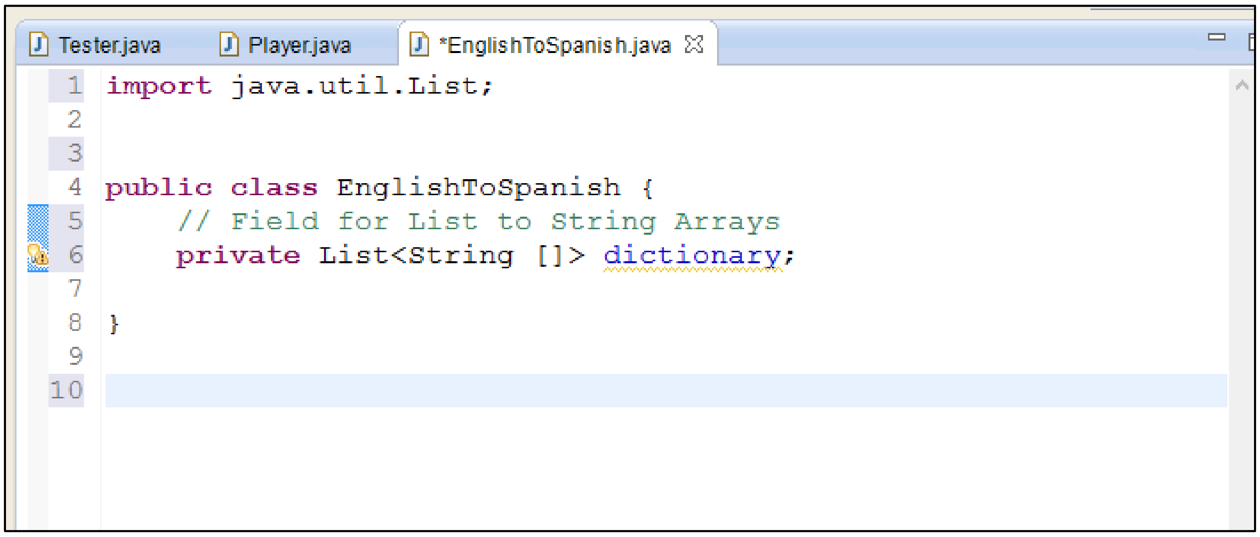
```

public class EnglishToSpanish {
    // Field for List to String Arrays
    private List<String []> dictionary;

}

```

List cannot be resolved to a type

11 quick fixes available:

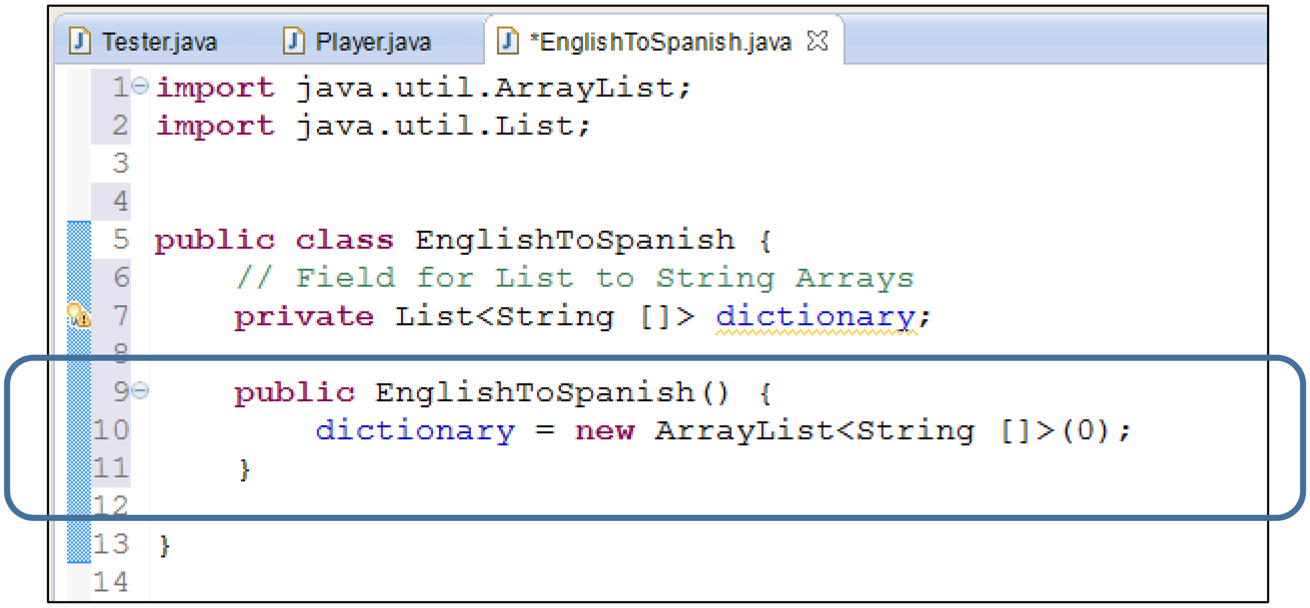
- Import 'List' (java.util)
- Import 'List' (java.awt)
- Create class 'List<T>'
- Create interface 'List<T>'
- Change to 'Line' (javax.sound.sampled)
- Change to 'Listener' (javax.xml.bind.Marshaller)
- Change to 'Listener' (javax.xml.bind.Unmarshaller)
- Change to 'ListPeer' (java.awt.peer)

The code should now look like this: (Note the addition of code on line 1). This is called an import statement and it brings in additional libraries of objects and functions.



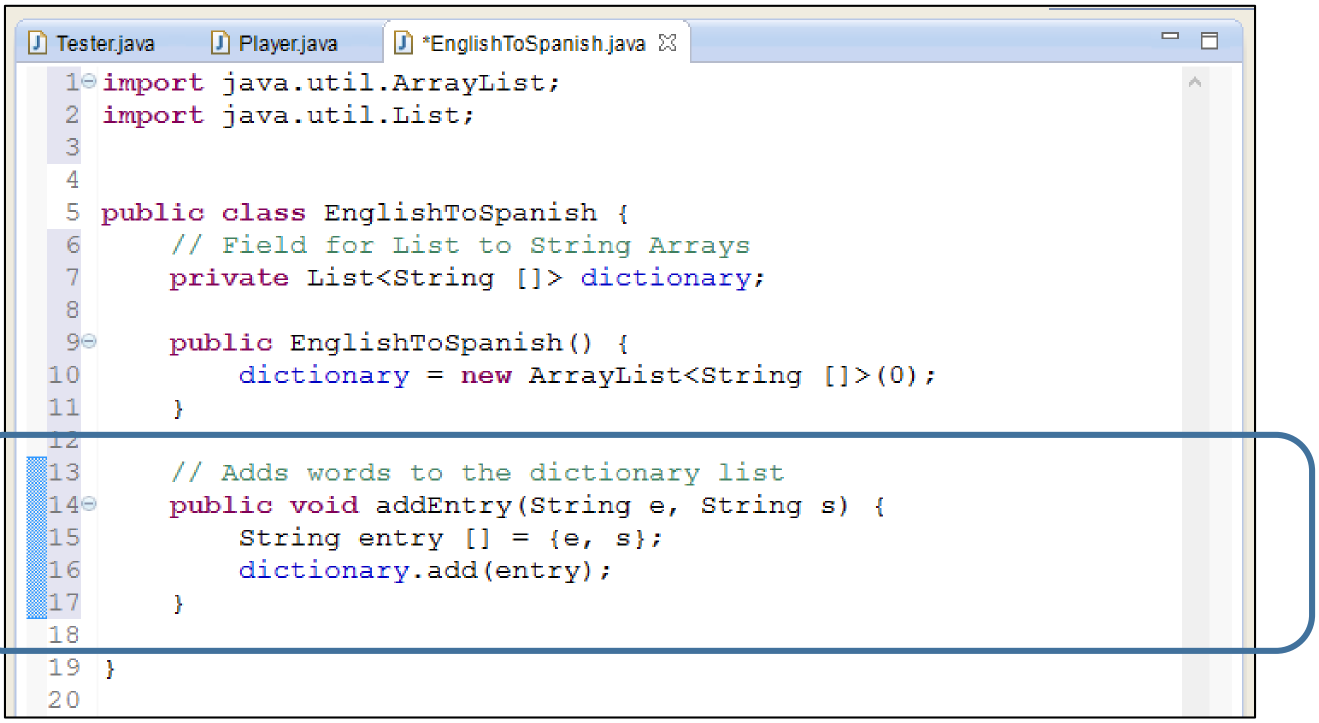
```
1  import java.util.List;
2
3
4  public class EnglishToSpanish {
5      // Field for List to String Arrays
6      private List<String []> dictionary;
7
8  }
9
10
```

30. Next we will write the Constructor. This is where we will initialize the List dictionary. You will have to hover and import ArrayList to remove the error.



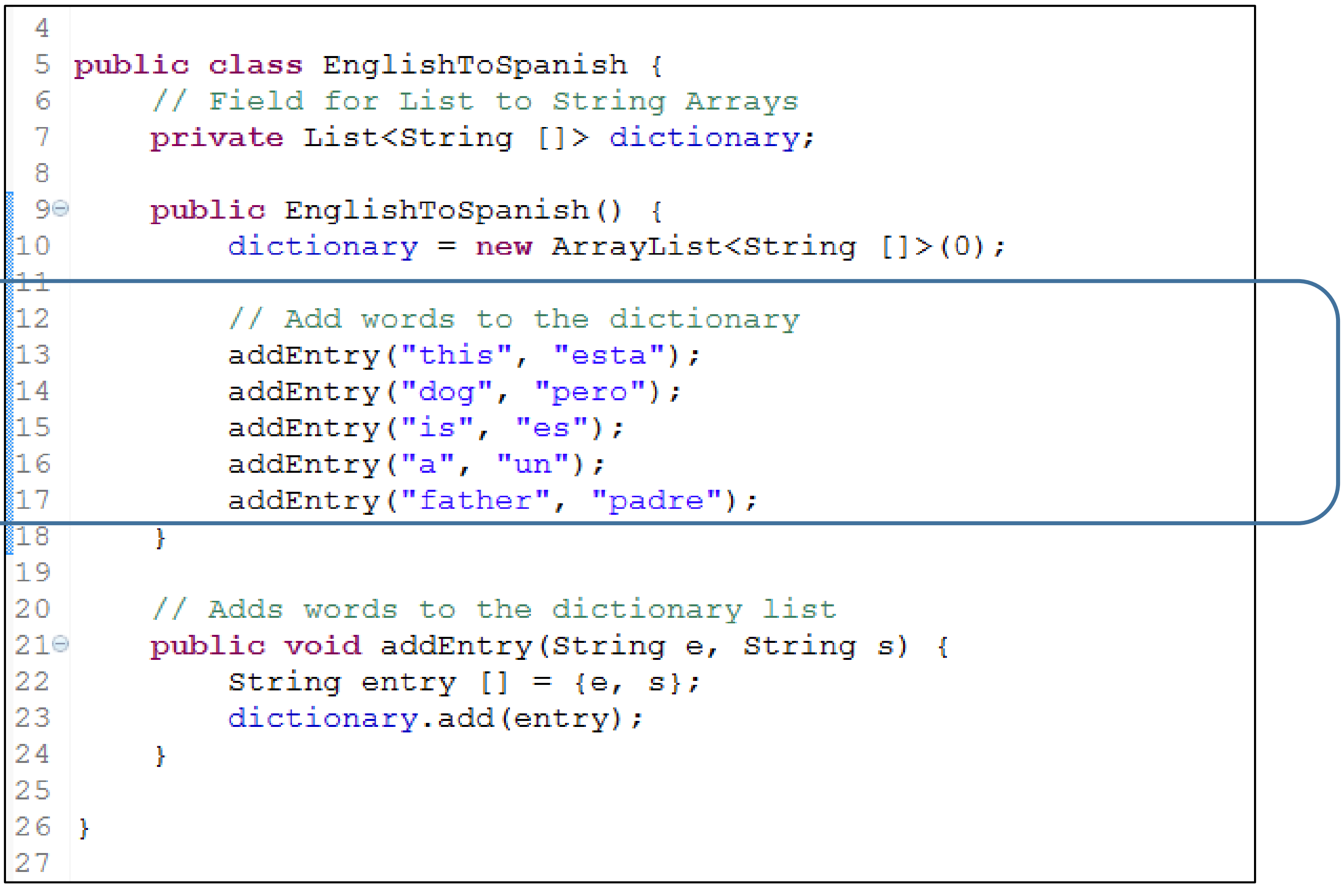
```
1  import java.util.ArrayList;
2  import java.util.List;
3
4
5  public class EnglishToSpanish {
6      // Field for List to String Arrays
7      private List<String []> dictionary;
8
9      public EnglishToSpanish() {
10         dictionary = new ArrayList<String []>(0);
11     }
12
13 }
14
```

31. We will now write the function that will allow the programmer to add new words to the dictionary object:



```java
import java.util.ArrayList;
import java.util.List;


public class EnglishToSpanish {
    // Field for List to String Arrays
    private List<String []> dictionary;

    public EnglishToSpanish() {
        dictionary = new ArrayList<String []>(0);
    }

    // Adds words to the dictionary list
    public void addEntry(String e, String s) {
        String entry [] = {e, s};
        dictionary.add(entry);
    }

}
```

32. Now that we have an 'addEntry' function, we can build a short list of English and Spanish words. Go back to the constructor and use 'addEntry' to add some words to the dictionary. (In actual Android programming, we would use a Google Play object. However, the Google Play service is not free and goes beyond the scope of this lesson. However, being able to manipulate Lists and Arrays is a core skill in programming in any language or environment).

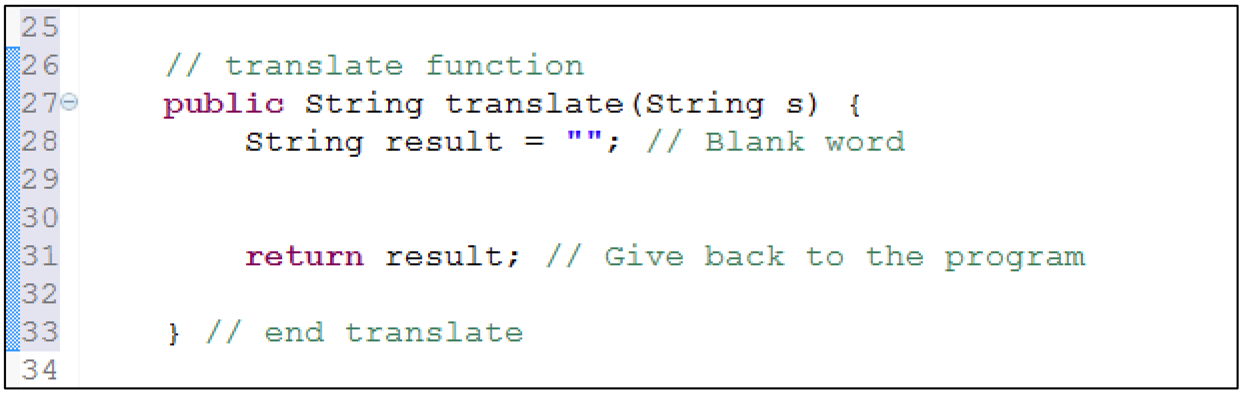


```

public class EnglishToSpanish {
    // Field for List to String Arrays
    private List<String []> dictionary;

    public EnglishToSpanish() {
        dictionary = new ArrayList<String []>(0);

        // Add words to the dictionary
        addEntry("this", "esta");
        addEntry("dog", "pero");
        addEntry("is", "es");
        addEntry("a", "un");
        addEntry("father", "padre");
    }

    // Adds words to the dictionary list
    public void addEntry(String e, String s) {
        String entry [] = {e, s};
        dictionary.add(entry);
    }

}

```

It will be helpful to think of the dictionary as having this structure:

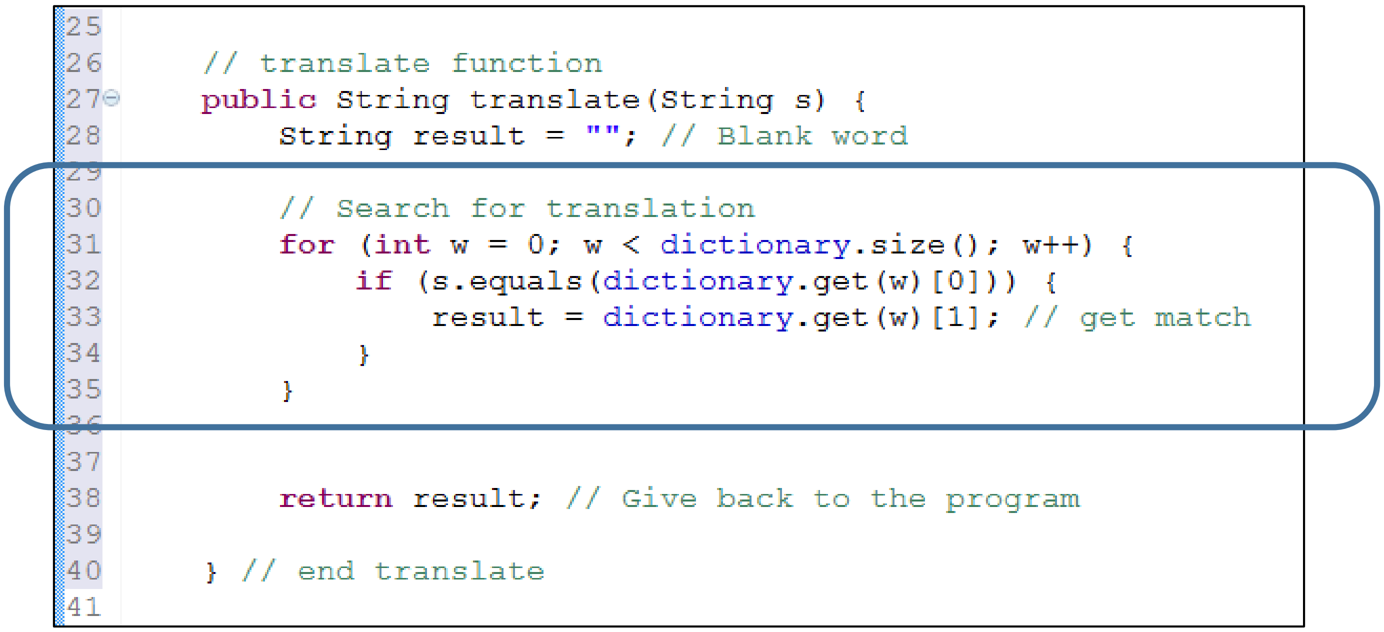
	[0]	**[1]**
dictionary.get(0)	this	esta
dictionary.get(1)	dog	pero
dictionary.get(2)	is	es
dictionary.get(3)	a	un
dictionary.get(4)	father	padre

33. We will now write the translate() function. Again, we are using Lists and arrays to be careful with the punctuation and other characters. Outline the function as shown below:



```java

    // translate function
    public String translate(String s) {
        String result = ""; // Blank word


        return result; // Give back to the program

    } // end translate

```

34. We will now write a for loop in the function that checks the incoming word 's' against the English words in the list. If there is a match, the result is set to the corresponding Spanish word.

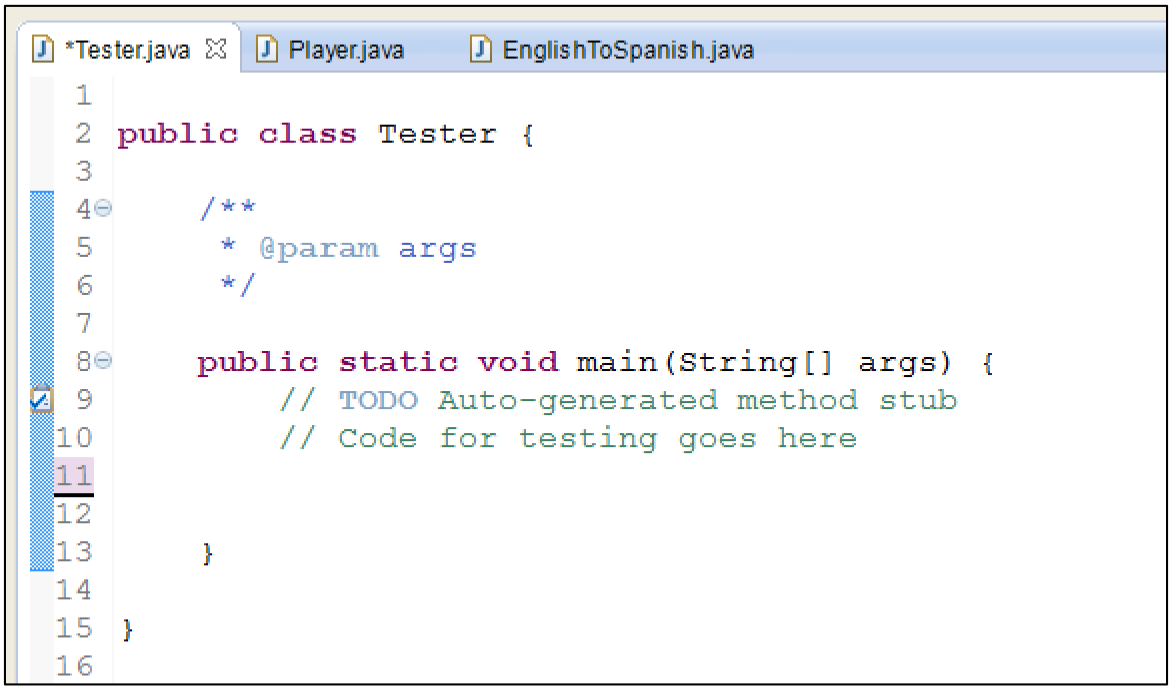


```java

    // translate function
    public String translate(String s) {
        String result = ""; // Blank word

        // Search for translation
        for (int w = 0; w < dictionary.size(); w++) {
            if (s.equals(dictionary.get(w)[0])) {
                result = dictionary.get(w)[1]; // get match
            }
        }


        return result; // Give back to the program

    } // end translate

```

35. Now if the word put into the translate function is not in the dictionary, we want to return the original word. Add this code to check for result being empty:

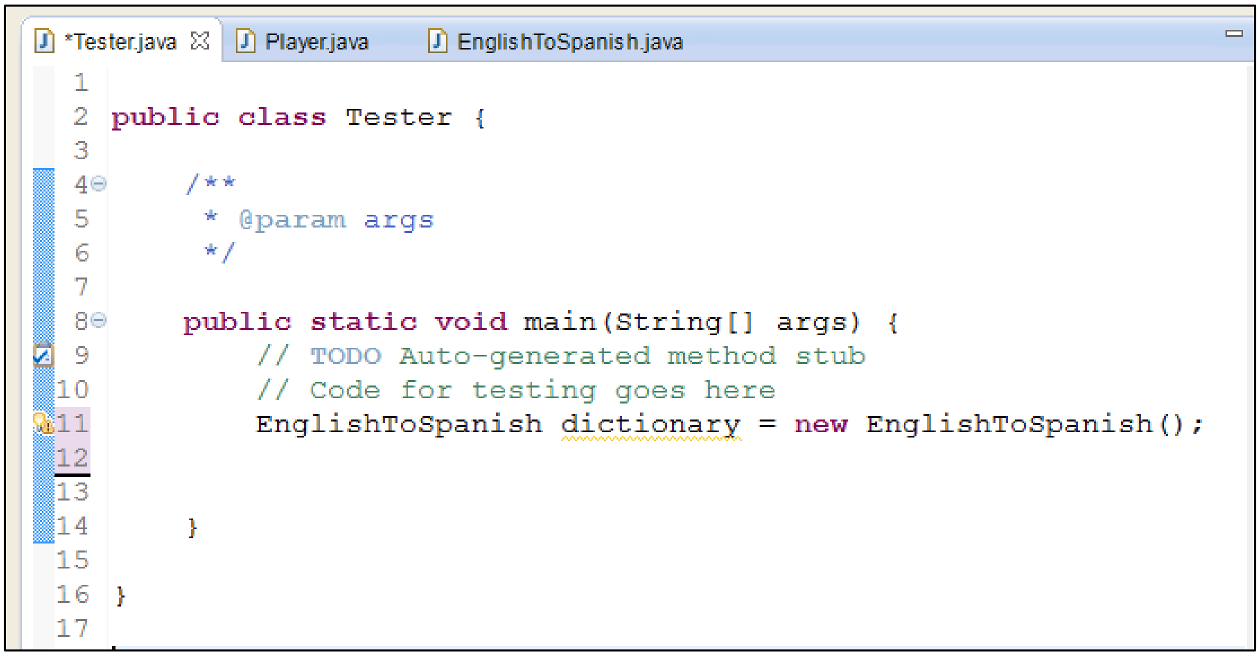
```
// translate function
public String translate(String s) {
    String result = ""; // Blank word

    // Search for translation
    for (int w = 0; w < dictionary.size(); w++) {
        if (s.equals(dictionary.get(w)[0])) {
            result = dictionary.get(w)[1]; // get match
        }
    }

    if (result.equals("")) {
        result = s; // set to incoming word
    }

    return result; // Give back to the program

} // end translate
```

36. You are finished with the EnglishToSpanish class. Now we want to run some tests. Go back to the Tester class and delete the code inside the main method.



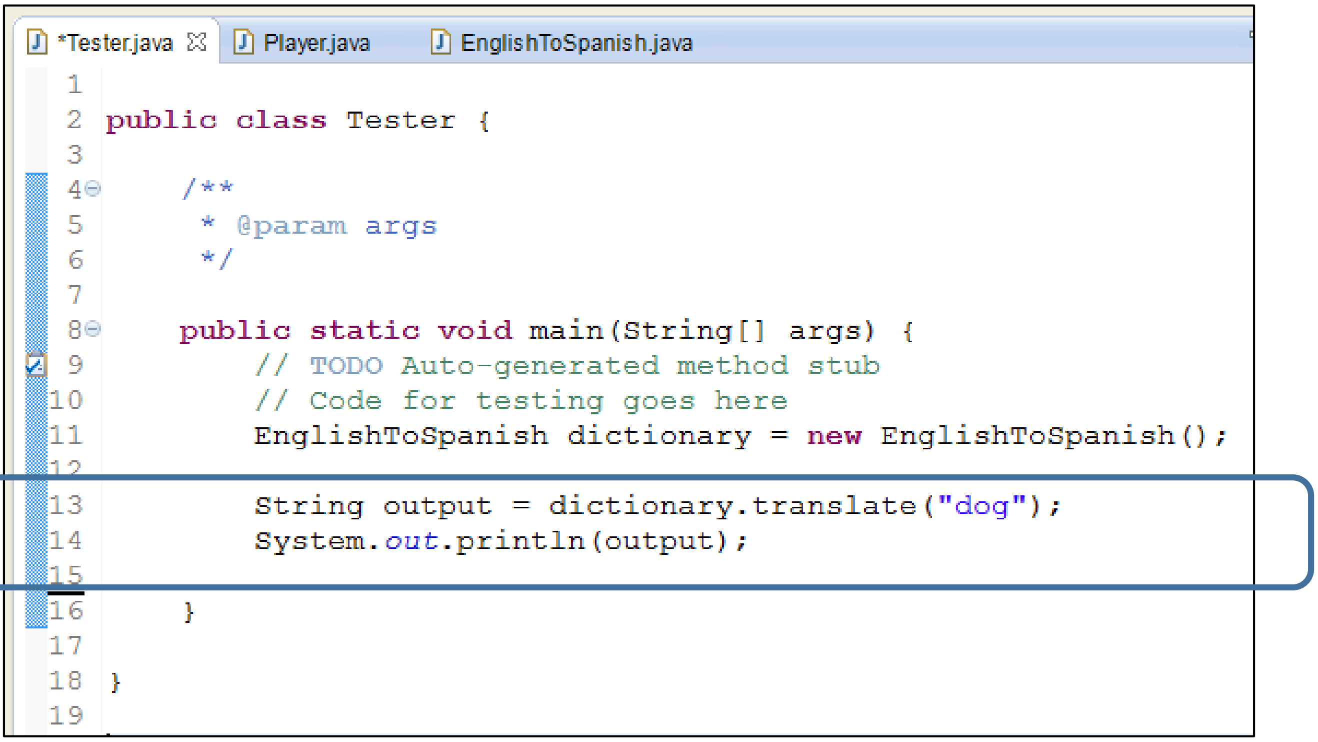
```

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here


    }

}

```

37. We will now add an instance of EnglishToSpanish:



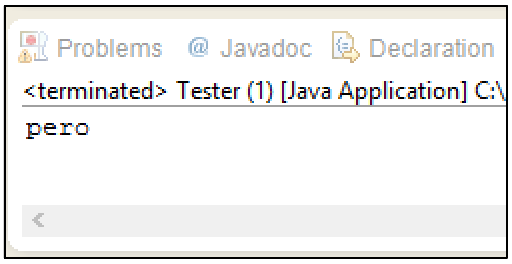
```

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        EnglishToSpanish dictionary = new EnglishToSpanish();


    }

}

```

38. We will now translate a word:



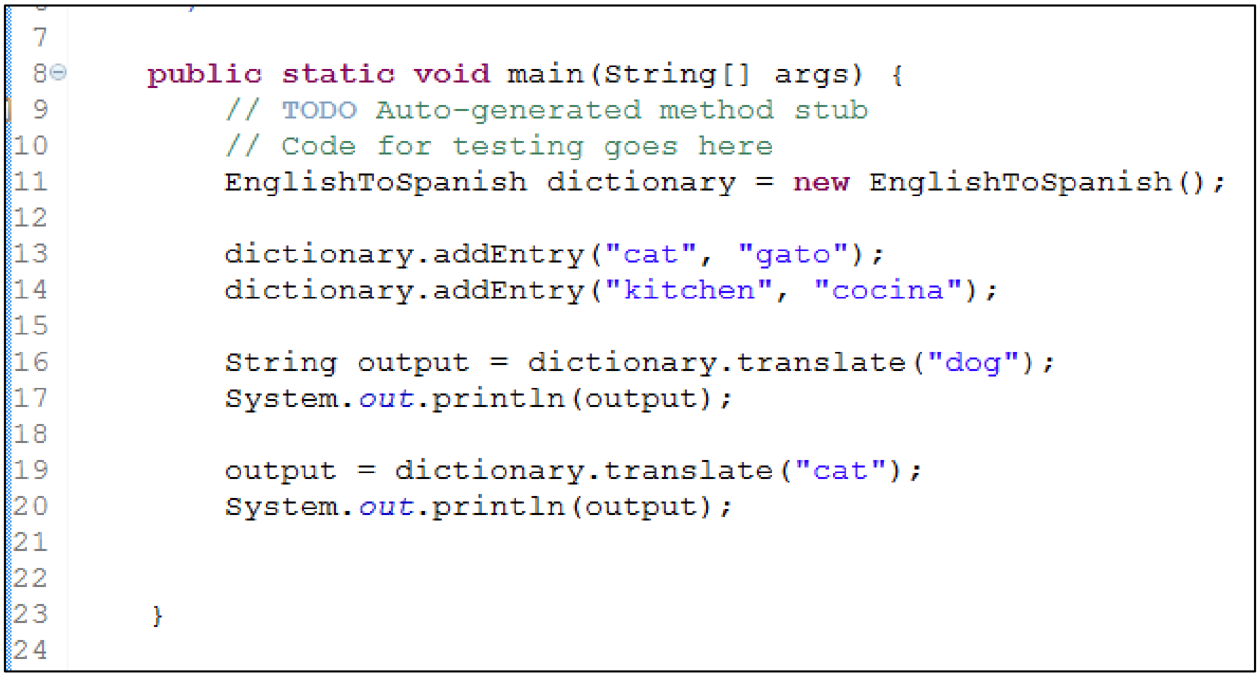
```

public class Tester {

    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        EnglishToSpanish dictionary = new EnglishToSpanish();

        String output = dictionary.translate("dog");
        System.out.println(output);

    }

}

```

Should output:


Problems @ Javadoc Declaration

```
<terminated> Tester (1) [Java Application] C:\
pero
```

39. We can add words using the .addEntry function:


```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Code for testing goes here
        EnglishToSpanish dictionary = new EnglishToSpanish();

        dictionary.addEntry("cat", "gato");
        dictionary.addEntry("kitchen", "cocina");

        String output = dictionary.translate("dog");
        System.out.println(output);

        output = dictionary.translate("cat");
        System.out.println(output);


    }
```

40. This completes the Test Platform exercise. During the course we will use the Test Platform to run short code ideas or work out class structures, inputs,and outputs.

41. Assignment:

a. Given the TestPlatform package and the Tester class we built, construct an additional class called 'Contact' . The 'Contact' class models an object that we would use to make an Address Book. Contact will have the following structure:

Contact
Fields: Private String lastname Private String firstname Private String email Private String cellphone Private String address
Constructor: public Contact(String lastname, String firstname)
Methods: Public String getName() (Hint = look up concatenating Strings in java . . .) Public String getEmail() Public String getCell() Public String getAddress() Public void setEmail(String e) Public void setCell(String c) Public void setAddress(String a) Public void setName(String last, String first)

b. In the Tester class, test the Contacts class by creating 2 instances and printing to the console examples of Name, email, cell, and address.

c. Create a new translator class of a language of your choice (EnglishToLatin . .). Model this class after the EnglishToSpanish and run tests of at least 5 words in the Tester class.

Scoring (Max 100 Points):

Category	Points
Complete Steps 1 to 40 in these directions -Tester class -Player class -English to Spanish class -Run tests for each class	70
Step 41.a -Create Contact Class	10
Step 41.b -Test Contact Class with 2 instances	10
Step 41.c -Create new translator class -Test 5 different translations	10