|  |  |
| --- | --- |
| Project Phase 2 Report | Course  Design Patterns  Course Instructor  Aisha Urooj  Group Members  M. Fawad Jawaid K112116  Shoaib Ahmed K112016  Zohaib Masood K112114  Mohammad Zohair K112181 |

Table of Contents

[Identified Bad Smells 2](#_Toc406190263)

[Long Method 2](#_Toc406190264)

[Refactoring Technique Used 2](#_Toc406190265)

[Data Class 2](#_Toc406190266)

[Refactoring Technique Used 2](#_Toc406190267)

[Comments 2](#_Toc406190268)

[Refactoring Technique Used 3](#_Toc406190269)

[Temporary Field 3](#_Toc406190270)

[Refactoring Technique Used 3](#_Toc406190271)

[Primitive Obsession 3](#_Toc406190272)

[Refactoring Technique Used 3](#_Toc406190273)

[Lines of Codes Changed 4](#_Toc406190274)

[For “Long Method” 4](#_Toc406190275)

[Replace Temp with Query 4](#_Toc406190276)

[Extract Method 5](#_Toc406190277)

[For “Data Class” Smell 8](#_Toc406190278)

[Encapsulate Field 8](#_Toc406190279)

[For “Comments” Smell 17](#_Toc406190280)

[Rename Method 17](#_Toc406190281)

[For “Temporary Field” Smell 18](#_Toc406190282)

[Introduce Null Object 18](#_Toc406190283)

[For “Primitive Obsession” Smell 20](#_Toc406190284)

[Replace Data Value with Object 20](#_Toc406190285)

[Application Design 21](#_Toc406190286)

[Before Refactoring 21](#_Toc406190287)

[Main Package 21](#_Toc406190288)

[Employee Package 22](#_Toc406190289)

[Visitor Package 23](#_Toc406190290)

[All Packages 24](#_Toc406190291)

[Complete Application Design 25](#_Toc406190292)

[After Refactoring (Changed Diagrams Only) 27](#_Toc406190293)

[Main Package 27](#_Toc406190294)

[Employee Package 28](#_Toc406190295)

[Work Breakdown Structure 29](#_Toc406190296)

[Members: 29](#_Toc406190297)

[Concluding Remarks 30](#_Toc406190298)

# Identified Bad Smells

## Long Method

It is a known fact in the world of programming that the longer a procedure is, the more difficult it is to understand. The length of the code is not the actual problem in this situation, but the semantic distance between what the method does and how it does it is. To detect this bad smell, a good technique is to look for comments in the code as they often indicate a semantic distance. If a comment tells what the following block of code is doing, then it can be replaced by a method whose name is based on the comment.

### Refactoring Technique Used

The refactoring technique we used to resolve the above bad smell is “Extract Method” and “Replace Temp with Query”. The first technique is used 2 times while the second one is used three times in our project.

## Data Class

Such classes which only hold fields, their getters and setters and nothing else are data classes. These classes just hold data and do not carry any behaviors with them. In early stages, most of the data classes have public fields, which is the actual disadvantage of this bad smell.

### Refactoring Technique Used

The refactoring technique we used to resolve the above bad smell is “Encapsulate Field”, using which we encapsulated all the public fields in the data classes. This technique is used seven times in our project as shown below.

## Comments

Comments are not a bad thing in programming but a lot of comments shows that the code is bad. There are three possibilities where comments are used against a bad code and considered a bad smell; one, when the programmer needs to tell what a block of code does, second, when the programmer wants to tell what a specific method does and third, when the programmer needs to state some rules about the required state of the system. These bad smells can be removed by extracting method, renaming method and introducing assertions. In short, a comment should tell why a code was written instead of what a code does.

### Refactoring Technique Used

The refactoring technique we used to resolve the above bad smell is “Rename Method” and “Extract Method”. The former technique is used once in our project while the latter one twice.

## Temporary Field

Sometimes a field of an object is set only in certain circumstances. Such objects are difficult to understand as an object is expected to need all of its fields. We can extract a new class including these fields to make object easier to understand. To create an alternative component for the object when the fields aren't valid, we can introduce a null object having unused fields as null.

### Refactoring Technique Used

The refactoring technique we used to resolve the above bad smell is “Introduce Null Object”. This technique is used thrice in our project as shown in the “Lines of Codes Changed” Section.

## Primitive Obsession

There are two categories of data types in every programming environment, one, primitive data types, such as “int”, “long”, which represents a single valued data, second, classes, which holds multiple primitive data types as well as some behaviors related to data. Most of the people are reluctant to make small classes, so when they need to do small tasks on small data, they use primitive data types and manipulate them. Here a small class should be made to keep this data and perform actions on this data.

### Refactoring Technique Used

The refactoring technique we used to resolve the above bad smell is “Replace Data Value with Object”. This technique is used once in our project.

# Lines of Codes Changed

## For “Long Method”

### Replace Temp with Query

#### Package: com.hci.virtualreceptionist

##### Class: CustomListView.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| Employee Employee = **new** Employee("Software Engineer","ABC",**false**);  EmployeeList.add(Employee);  Employee = **new** Employee("Tester","DEF",**true**);  EmployeeList.add(Employee);  Employee = **new** Employee("QA","GHI",**false**);  EmployeeList.add(Employee); | EmployeeList.add(**new** Employee ("Software Engineer","ABC",**false**));  EmployeeList.add(**new** Employee ("Tester","DEF",**true**));  EmployeeList.add(**new** Employee ("QA","GHI",**false**)); |

##### Class: LoginActivity.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| DialogBox frg1 = **new** LoginDialogFragment();  android.app.FragmentManager fm = getFragmentManager();  frg1.show(fm, ""); | **new** LoginDialogFragment().show(getFragmentManager(), ""); |
| Intent signupIntent = **new** Intent(getApplicationContext(),  SignupActivity.**class**); startActivity(signupIntent); | startActivity(**new** Intent(getApplicationContext(),SignupActivity.**class**);  ); |

##### Class: CustomListView.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| Employee Employee = **new** Employee("Software Engineer","ABC",**false**);  EmployeeList.add(Employee);  Employee = **new** Employee("Tester","DEF",**true**);  EmployeeList.add(Employee);  Employee = **new** Employee("QA","GHI",**false**);  EmployeeList.add(Employee); | EmployeeList.add(**new** Employee ("Software Engineer","ABC",**false**));  EmployeeList.add(**new** Employee ("Tester","DEF",**true**));  EmployeeList.add(**new** Employee ("QA","GHI",**false**)); |

### Extract Method

#### Package: com.hci.virtualreceptionist

##### Class: CustomListView.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| dataAdapter = **new** MyCustomAdapter(cont, R.layout.***custom\_listview***, EmployeeList);  ListView listView = (ListView) ((Activity)cont).findViewById(R.id.***listView1***);  listView.setAdapter(dataAdapter);    listView.setOnItemClickListener(**new** OnItemClickListener() {  **public** **void** onItemClick(AdapterView<?> parent, View view,  **int** position, **long** id) {  Employee Employee = (Employee) parent.getItemAtPosition(position);  Toast.*makeText*(cont,  "Clicked on Row: " + Employee.getName(),  Toast.***LENGTH\_LONG***).show();  }  }); | **private** **void** settingListAdapter(ArrayList<Employee> EmployeeList)  {    dataAdapter = **new** MyCustomAdapter(cont,  R.layout.***custom\_listview***, EmployeeList);  ListView listView = (ListView) ((Activity)cont).findViewById(R.id.***listView1***);  listView.setAdapter(dataAdapter);  onClickListItem(listView);    }    **private** **void** onClickListItem(ListView listView)  {  listView.setOnItemClickListener(**new** OnItemClickListener() {  **public** **void** onItemClick(AdapterView<?> parent, View view, **int** position, **long** id) {    Employee Employee = (Employee) parent.getItemAtPosition(position);  Toast.*makeText*(cont,  "Clicked on Row: " + Employee.getName(),  Toast.***LENGTH\_LONG***).show();  }  });    } |
| LayoutInflater vi = (LayoutInflater)cont.getSystemService(  Context.***LAYOUT\_INFLATER\_SERVICE***);  convertView = vi.inflate(R.layout.***custom\_listview***, **null**);    holder = **new** ViewHolder();  holder.code = (TextView) convertView.findViewById(R.id.***code***);  holder.name = (CheckBox) convertView.findViewById(R.id.***checkBox1***);  convertView.setTag(holder);    holder.name.setOnClickListener( **new** View.OnClickListener() {  **public** **void** onClick(View v) {  CheckBox cb = (CheckBox) v ;  Employee Employee = (Employee) cb.getTag();  Toast.*makeText*(cont,  "Clicked on Checkbox: " + cb.getText() +  " is " + cb.isChecked(),  Toast.***LENGTH\_LONG***).show();  Employee.setSelected(cb.isChecked());  }  });  }  **else** {  holder = (ViewHolder) convertView.getTag();  } | **private** ViewHolder listItemDecorator(View convertView) {  ViewHolder holder = **null**;    **if** (convertView == **null**) {  LayoutInflater vi = (LayoutInflater)cont.getSystemService(  Context.***LAYOUT\_INFLATER\_SERVICE***);  convertView = vi.inflate(R.layout.***custom\_listview***, **null**);    holder = **new** ViewHolder();  holder.code = (TextView) convertView.findViewById(R.id.***code***);  holder.name = (CheckBox) convertView.findViewById(R.id.***checkBox1***);  convertView.setTag(holder);    holder.name.setOnClickListener( **new** View.OnClickListener() {  **public** **void** onClick(View v) {  CheckBox cb = (CheckBox) v ;  Employee Employee = (Employee) cb.getTag();  Toast.*makeText*(cont,  "Clicked on Checkbox: " + cb.getText() +  " is " + cb.isChecked(),  Toast.***LENGTH\_LONG***).show();  Employee.setSelected(cb.isChecked());  }  });  }  **else** {  holder = (ViewHolder) convertView.getTag();  }  **return** holder;  } |

##### Class: LoginActivity.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);    setContentView(R.layout.***activity\_login***);  loginBtn = (Button) findViewById(R.id.***loginbutton***);  visitorBtn = (Button) findViewById(R.id.***visitorbutton***);  signupBtn = (Button) findViewById(R.id.***signupbutton***);    loginBtn.setOnClickListener(loginBtnListener);  visitorBtn.setOnClickListener(visitorBtnListener);  signupBtn.setOnClickListener(signupBtnListener);  } | **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  initUI();  }  **private** **void** initUI() {  setContentView(R.layout.***activity\_login***);    loginBtn = (Button) findViewById(R.id.***loginbutton***);  visitorBtn = (Button) findViewById(R.id.***visitorbutton***);  signupBtn = (Button) findViewById(R.id.***signupbutton***);    loginBtn.setOnClickListener(loginBtnListener);  visitorBtn.setOnClickListener(visitorBtnListener);  signupBtn.setOnClickListener(signupBtnListener);  } |

## For “Data Class” Smell

### Encapsulate Field

#### Package: com.hci.virtualreceptionist

##### Class: Appointment.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package com.hci.virtualreceptionist;**  **import com.hci.virtualreceptionist.interfaces.Observer;**  **import com.hci.virtualreceptionist.interfaces.Subject;**  **public class Appointment implements Subject{**  **Observer obs;**  **String status;**    **String subject;**  **String dateTime;**  **Employee appointee; //refactored, using Replace Data Value with Object**    **private static Appointment appointment;**    **private Appointment(String sub, String date, Employee emp) {**  **this.subject = sub;**  **this.dateTime = date;**  **this.appointee = emp;**  **}**    **public String getSubject() { return subject;**  **}**    **public String getDateTime() {**  **return dateTime;**  **}**    **public Employee getAppointee() {**  **return appointee;**  **}**    **public void setSubject(String subject) {**  **this.subject = subject;**  **}**    **public void setDateTime(String date) {**  **this.dateTime = date;**  **}**    **public void setAppointee(Employee app) {**  **this.appointee = app;**  **}**      **public static Appointment getInstance(String sub, String date, Employee emp) {**    **if(appointment == null) {**  **appointment = new Appointment(sub, date, emp);**  **}**    **return appointment;**  **}**      **@Override**  **public void registerObserver(Observer o) { //Observer Pattern**  **this.obs = o;**  **}**  **@Override**  **public void removeObserver() {**  **this.obs = null;**  **}**  **@Override**  **public void notifyObserver() {**  **this.obs.update(this.status);**  **}**  **}** | **package com.hci.virtualreceptionist;**  **import com.hci.virtualreceptionist.interfaces.Observer;**  **import com.hci.virtualreceptionist.interfaces.Subject;**  **public class Appointment implements Subject{**  **Observer obs;**  **private String status;**    **private String subject;**  **private String dateTime;**  **Employee appointee;**    **private static Appointment appointment;**    **private Appointment(String sub, String date, Employee emp) {**  **this.subject = sub;**  **this.dateTime = date;**  **this.appointee = emp;**  **}**    **public String getSubject() {**  **return subject;**  **}**    **public String getDateTime() {**  **return dateTime;**  **}**    **public Employee getAppointee() {**  **return appointee;**  **}**    **public void setSubject(String subject) {**  **this.subject = subject;**  **}**    **public void setDateTime(String date) {**  **this.dateTime = date;**  **}**    **public void setAppointee(Employee app) {**  **this.appointee = app;**  **}**      **public static Appointment getInstance(String sub, String date, Employee emp) {**    **if(appointment == null) {**  **appointment = new Appointment(sub, date, emp);**  **}**    **return appointment;**  **}**      **@Override**  **public void registerObserver(Observer o) {**  **this.obs = o;**  **}**  **@Override**  **public void removeObserver() {**  **this.obs = null;**  **}**  **@Override**  **public void notifyObserver() {**  **this.obs.update(this.status);**  **}**  **}** |

##### Class: Employee.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist;  **public** **class** Employee {  **public** String designation = **null**;  **public** String name = **null**;  **public** **boolean** selected = **false**;    **public** Employee(String designation, String name, **boolean** selected) {  **super**();  **this**.designation = designation;  **this**.name = name;  **this**.selected = selected;  }    **public** String getDesignation() { **return** designation;  }  **public** **void** setCode(String designation) {  **this**.designation = designation;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** **boolean** isSelected() {  **return** selected;  }  **public** **void** setSelected(**boolean** selected) {  **this**.selected = selected;  }  } | **package** com.hci.virtualreceptionist;  **public** **class** Employee {  **private** String designation = **null**;  **private** String name = **null**;  **boolean** selected = **false**;    **public** Employee(String designation, String name, **boolean** selected) {  **super**();  **this**.designation = designation;  **this**.name = name;  **this**.selected = selected;  }    **public** String getDesignation() {  **return** designation;  }  **public** **void** setCode(String designation) {  **this**.designation = designation;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** **boolean** isSelected() {  **return** selected;  }  **public** **void** setSelected(**boolean** selected) {  **this**.selected = selected;  }  } |

##### Class: Visitor.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist;  **import** com.hci.virtualreceptionist.interfaces.Observer;  **public** **class** Visitor **implements** Observer{    **public** String name = **null**;  **public** String cnic = **null**;  **public** String appointmentStatus = "Pending";    **boolean** selected = **false**;    **public** Visitor(String name, String cnic) {  **super**();  **this**.name = name;  **this**.cnic = cnic;  }    **public** String getName() {  **return** name;  }    **public** **void** setName(String name) {  **this**.name = name;  }    **public** String getCnic() {  **return** cnic;  }    **public** **void** setCnic(String nic) {  **this**.cnic = nic;  }    **public** **boolean** isSelected() {  **return** selected;  }    **public** **void** setSelected(**boolean** selected) {  **this**.selected = selected;  }    **public** **void** update(String status) {  appointmentStatus = status;  }    } | **package** com.hci.virtualreceptionist;  **import** com.hci.virtualreceptionist.interfaces.Observer;  **public** **class** Visitor **implements** Observer{    **private** String name = **null**;  **private** String cnic = **null**;  **private** String appointmentStatus = "Pending";    **boolean** selected = **false**;    **public** Visitor(String name, String cnic) {  **super**();  **this**.name = name;  **this**.cnic = cnic;  }    **public** String getName() {  **return** name;  }    **public** **void** setName(String name) {  **this**.name = name;  }    **public** String getCnic() {  **return** cnic;  }    **public** **void** setCnic(String nic) {  **this**.cnic = nic;  }    **public** **boolean** isSelected() {  **return** selected;  }    **public** **void** setSelected(**boolean** selected) {  **this**.selected = selected;  }    **public** **void** update(String status) {  appointmentStatus = status;  }    } |

#### Package: com.hci.virtualreceptionist.employee.controller

##### Class: Document.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** Document {  **public** String name;  **public** String filepath;  **public** Document(String etitle, String efile){  **this**.name = etitle;  **this**.filepath = efile;  }  **public** String getName() { **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** String getFile() {  **return** filepath;  }  **public** **void** setFile(String file) {  **this**.filepath = file;  }  } | **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** Document {  **private** String name;  **private** String filepath;  **public** Document(String etitle, String efile){  **this**.name = etitle;  **this**.filepath = efile;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** String getFile() {  **return** filepath;  }  **public** **void** setFile(String file) {  **this**.filepath = file;  }  } |

##### Class: EMenuItem.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** EMenuItem {  **public** String title;  **public** **int** icon;  **public** String count = "0";  **public** **boolean** isCounterVisible = **false**;    **public** EMenuItem(){}  **public** EMenuItem(String title, **int** icon){  **this**.title = title;  **this**.icon = icon;  }    **public** EMenuItem(String title, **int** icon, **boolean** isCounterVisible, String count){  **this**.title = title;  **this**.icon = icon;  **this**.isCounterVisible = isCounterVisible;  **this**.count = count;  }    **public** String getTitle(){ **return** **this**.title;  }    **public** **int** getIcon(){  **return** **this**.icon;  }    **public** String getCount(){  **return** **this**.count;  }    **public** **boolean** getCounterVisibility(){  **return** **this**.isCounterVisible;  }    **public** **void** setTitle(String title){  **this**.title = title;  }    **public** **void** setIcon(**int** icon){  **this**.icon = icon;  }    **public** **void** setCount(String count){  **this**.count = count;  }    **public** **void** setCounterVisibility(**boolean** isCounterVisible){  **this**.isCounterVisible = isCounterVisible;  }  } | **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** EMenuItem {  **private** String title;  **private** **int** icon;  **private** String count = "0";    **private** **boolean** isCounterVisible = **false**;    **public** EMenuItem(){}  **public** EMenuItem(String title, **int** icon){  **this**.title = title;  **this**.icon = icon;  }    **public** EMenuItem(String title, **int** icon, **boolean** isCounterVisible, String count){  **this**.title = title;  **this**.icon = icon;  **this**.isCounterVisible = isCounterVisible;  **this**.count = count;  }    **public** String getTitle(){  **return** **this**.title;  }    **public** **int** getIcon(){  **return** **this**.icon;  }    **public** String getCount(){  **return** **this**.count;  }    **public** **boolean** getCounterVisibility(){  **return** **this**.isCounterVisible;  }    **public** **void** setTitle(String title){  **this**.title = title;  }    **public** **void** setIcon(**int** icon){  **this**.icon = icon;  }    **public** **void** setCount(String count){  **this**.count = count;  }    **public** **void** setCounterVisibility(**boolean** isCounterVisible){  **this**.isCounterVisible = isCounterVisible;  }  } |

##### Class: Meeting.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** Meeting {  **public** String title;  **public** String agenda;  **public** String date;  **public** String time;    **public** Meeting(String etitle, String eagenda, String edate, String etime){  **this**.title = etitle;  **this**.agenda = eagenda;  **this**.date = edate;  **this**.time = etime;  }  **public** String getTitle() { **return** title;  }  **public** **void** setTitle(String title) {  **this**.title = title;  }    **public** String getAgenda() {  **return** agenda;  }  **public** **void** setAgenda(String agenda) {  **this**.agenda = agenda;  }    **public** String getDate() {  **return** date;  }  **public** **void** setDate(String date) {  **this**.date = date;  }    **public** String getTime() {  **return** time;  }  **public** **void** setTime(String time) {  **this**.time = time;  }  } | **package** com.hci.virtualreceptionist.employee.controller;  **public** **class** Meeting {  **private** String title;  **private** String agenda;  **private** String date;  **private** String time;    **public** Meeting(String etitle, String eagenda, String edate, String etime){  **this**.title = etitle;  **this**.agenda = eagenda;  **this**.date = edate;  **this**.time = etime;  }  **public** String getTitle() {  **return** title;  }  **public** **void** setTitle(String title) {  **this**.title = title;  }    **public** String getAgenda() {  **return** agenda;  }  **public** **void** setAgenda(String agenda) {  **this**.agenda = agenda;  }    **public** String getDate() {  **return** date;  }  **public** **void** setDate(String date) {  **this**.date = date;  }    **public** String getTime() {  **return** time;  }  **public** **void** setTime(String time) {  **this**.time = time;  }  } |

##### Class: VMenuItem.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **package** com.hci.virtualreceptionist.visitor.controller;  **public** **class** VMenuItem {  **public** String title;  **public** **int** icon;  **public** String count = "0";  **public** **boolean** isCounterVisible = **false**;    **public** VMenuItem(){}  **public** VMenuItem(String title, **int** icon){  **this**.title = title;  **this**.icon = icon;  }    **public** VMenuItem(String title, **int** icon, **boolean** isCounterVisible, String count){  **this**.title = title;  **this**.icon = icon;  **this**.isCounterVisible = isCounterVisible;  **this**.count = count;  }    **public** String getTitle(){ **return** **this**.title;  }    **public** **int** getIcon(){  **return** **this**.icon;  }    **public** String getCount(){  **return** **this**.count;  }    **public** **boolean** getCounterVisibility(){  **return** **this**.isCounterVisible;  }    **public** **void** setTitle(String title){  **this**.title = title;  }    **public** **void** setIcon(**int** icon){  **this**.icon = icon;  }    **public** **void** setCount(String count){  **this**.count = count;  }    **public** **void** setCounterVisibility(**boolean** isCounterVisible){  **this**.isCounterVisible = isCounterVisible;  }  } | **package** com.hci.virtualreceptionist.visitor.controller;  **public** **class** VMenuItem {  **private** String title;  **private** **int** icon;  **private** String count = "0";  r  **private** **boolean** isCounterVisible = **false**;    **public** VMenuItem(){}  **public** VMenuItem(String title, **int** icon){  **this**.title = title;  **this**.icon = icon;  }    **public** VMenuItem(String title, **int** icon, **boolean** isCounterVisible, String count){  **this**.title = title;  **this**.icon = icon;  **this**.isCounterVisible = isCounterVisible;  **this**.count = count;  }    **public** String getTitle(){  **return** **this**.title;  }    **public** **int** getIcon(){  **return** **this**.icon;  }    **public** String getCount(){  **return** **this**.count;  }    **public** **boolean** getCounterVisibility(){  **return** **this**.isCounterVisible;  }    **public** **void** setTitle(String title){  **this**.title = title;  }    **public** **void** setIcon(**int** icon){  **this**.icon = icon;  }    **public** **void** setCount(String count){  **this**.count = count;  }    **public** **void** setCounterVisibility(**boolean** isCounterVisible){  **this**.isCounterVisible = isCounterVisible;  }  } |

## For “Comments” Smell

### Rename Method

#### Package: com.hci.virtualreceptionist.employee.views

##### Class: FragmentViewMeeting.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| //setting list adapters    **private** **void** setAdapters() {  lang\_adapter = **new**  MeetingAdapter(getActivity(),  R.layout.***meeting\_listitem***, GlobalVariables.*meetings*);  setListAdapter(lang\_adapter);  } | **private** **void** setListAdapters() {  lang\_adapter = **new** MeetingAdapter(getActivity(),  R.layout.***meeting\_listitem***, GlobalVariables.*meetings*);  setListAdapter(lang\_adapter);  } |

## For “Temporary Field” Smell

### Introduce Null Object

#### Package: com.hci.virtualreceptionist.employee.controller

##### Class: DocumentAdapter.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **if** (lang != **null**) {  TextView nameHeading = (TextView) v.findViewById(R.id.***txtName***);    ImageButton fileBtn = (ImageButton) v.findViewById(R.id.***fileButton***);    langPopularity = (TextView) v.findViewById(R.id.***txtAgenda***);    **if** (nameHeading != **null**){  nameHeading.setText(lang.getName());  }  } | **if** (!((NullDocument) lang).isNull()) {  TextView nameHeading = (TextView) v.findViewById(R.id.***txtName***);    ImageButton fileBtn = (ImageButton) v.findViewById(R.id.***fileButton***);    langPopularity = (TextView) v.findViewById(R.id.***txtAgenda***);    **if** (nameHeading != **null**){  nameHeading.setText(lang.getName());  } } |

#### Package: com.hci.virtualreceptionist.employee.views

##### Class: EMenuActivity.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **if**(fragment != **null**)  {  ft.replace(R.id.***activity\_main\_content\_fragment***, fragment);  ft.commit();  } | **if**(!((NullFragment) fragment).isNull()) {  ft.replace(R.id.***activity\_main\_content\_fragment***, fragment);  ft.commit();  } |

#### Package: com.hci.virtualreceptionist.visitor.views

##### Class: VMenuActivity.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| **if**(fragment != **null**)  {  ft.replace(R.id.***activity\_main\_content\_fragment***, fragment);  ft.commit();  } | **if**(!((NullFragment) fragment).isNull()) {  ft.replace(R.id.***activity\_main\_content\_fragment***, fragment);  ft.commit();  } |

## For “Primitive Obsession” Smell

### Replace Data Value with Object

#### Package: com.hci.virtualreceptionist

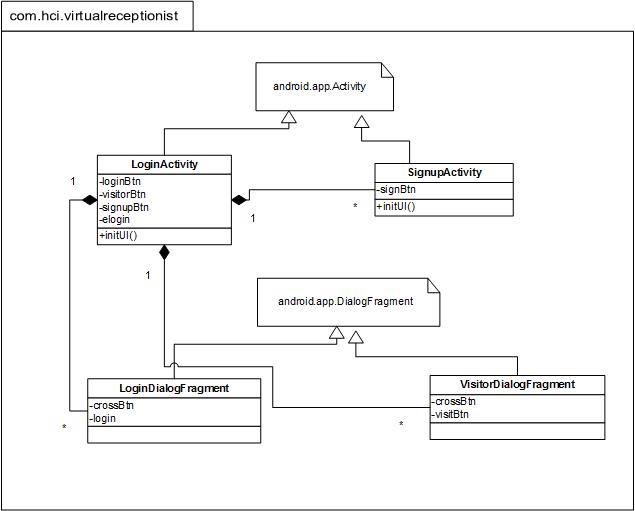
##### Class: Appointment.java

|  |  |
| --- | --- |
| **Code before Refactoring** | **Code after Refactoring** |
| String appointee;  // Employee's Name | Employee appointee;  // Employee Object |

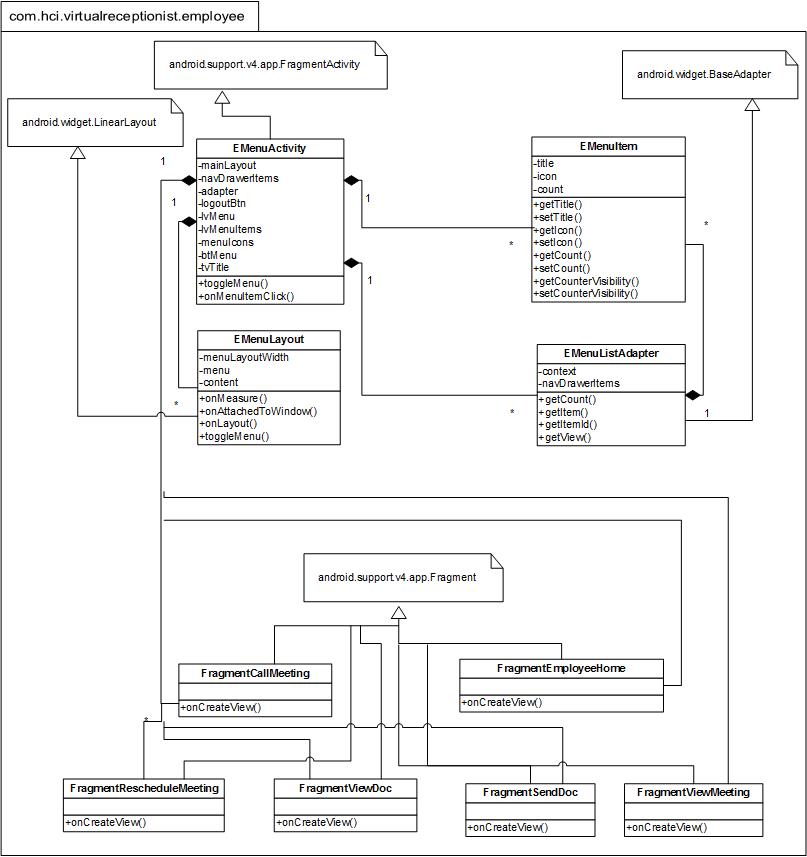
# Application Design

## Before Refactoring

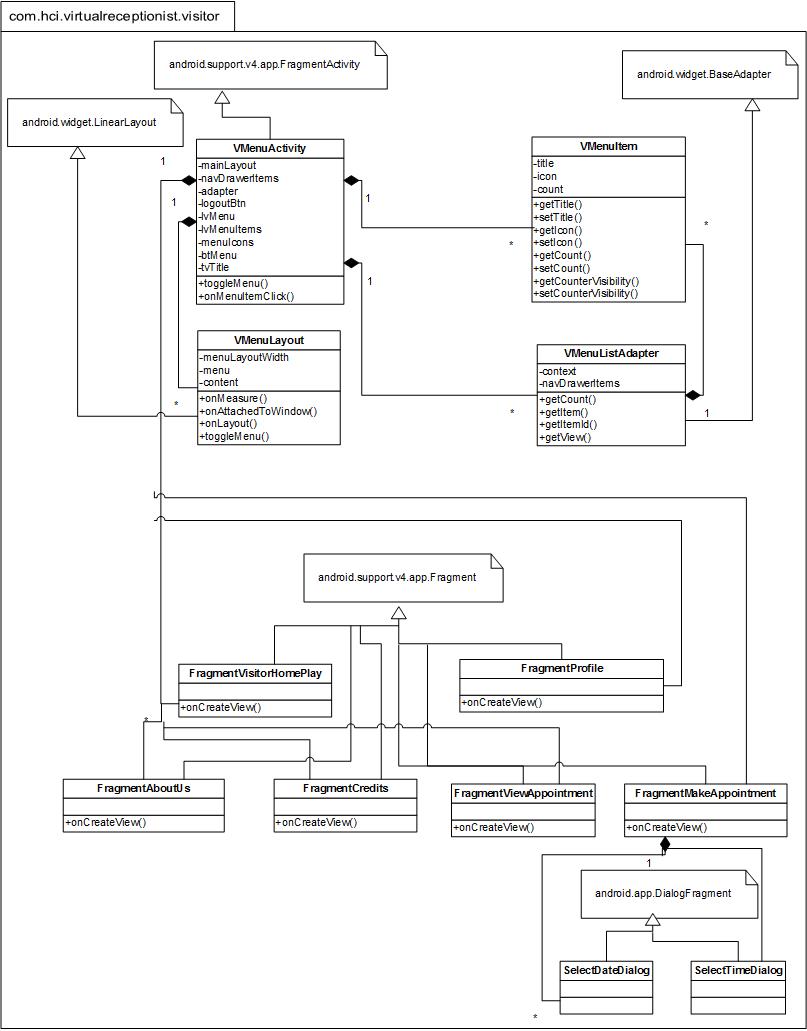
### Main Package



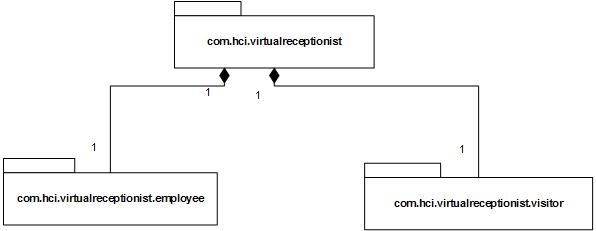
### Employee Package



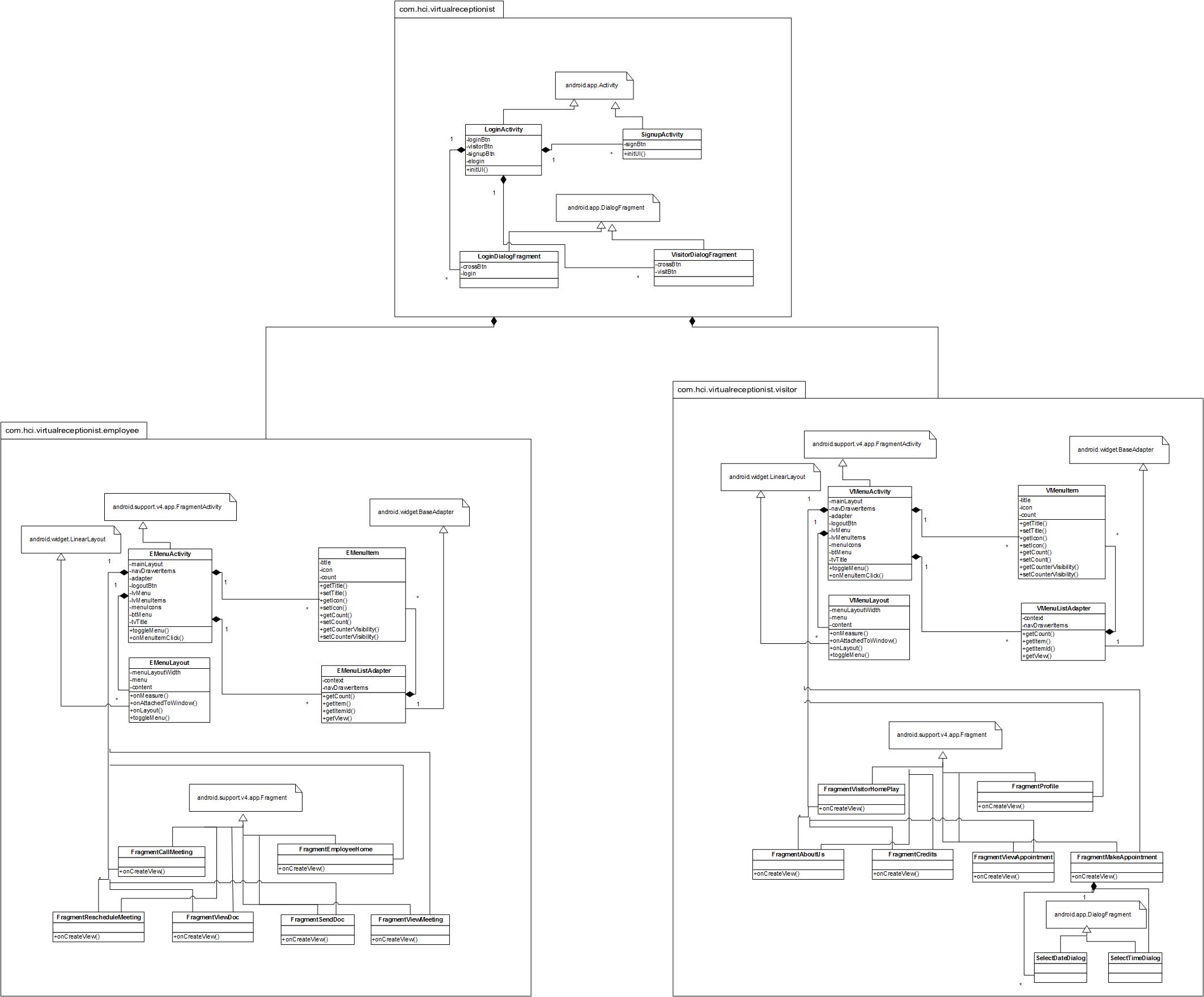
### Visitor Package



### All Packages

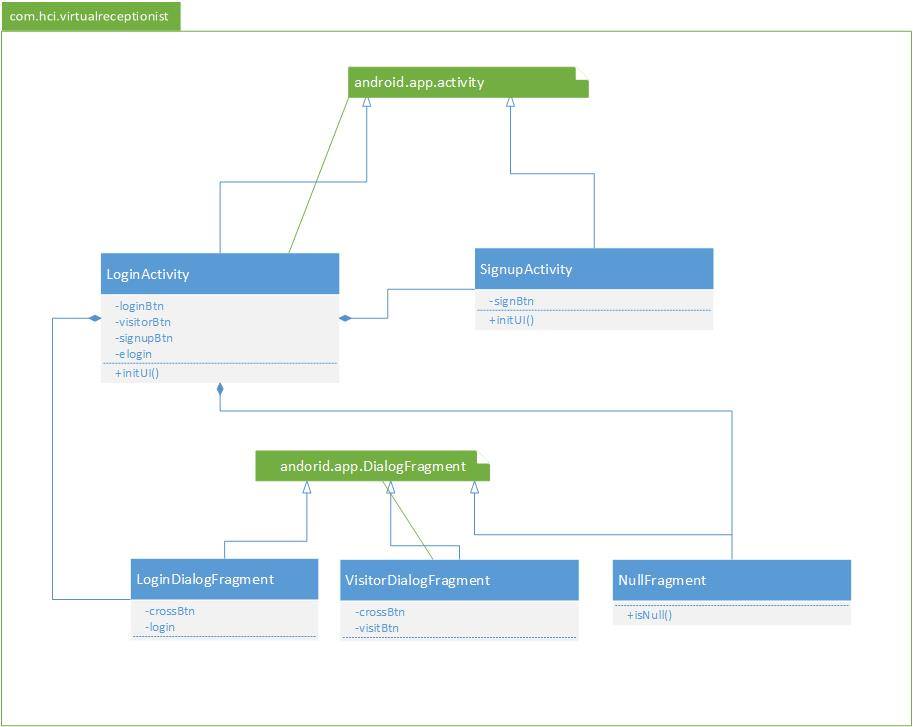


### Complete Application Design

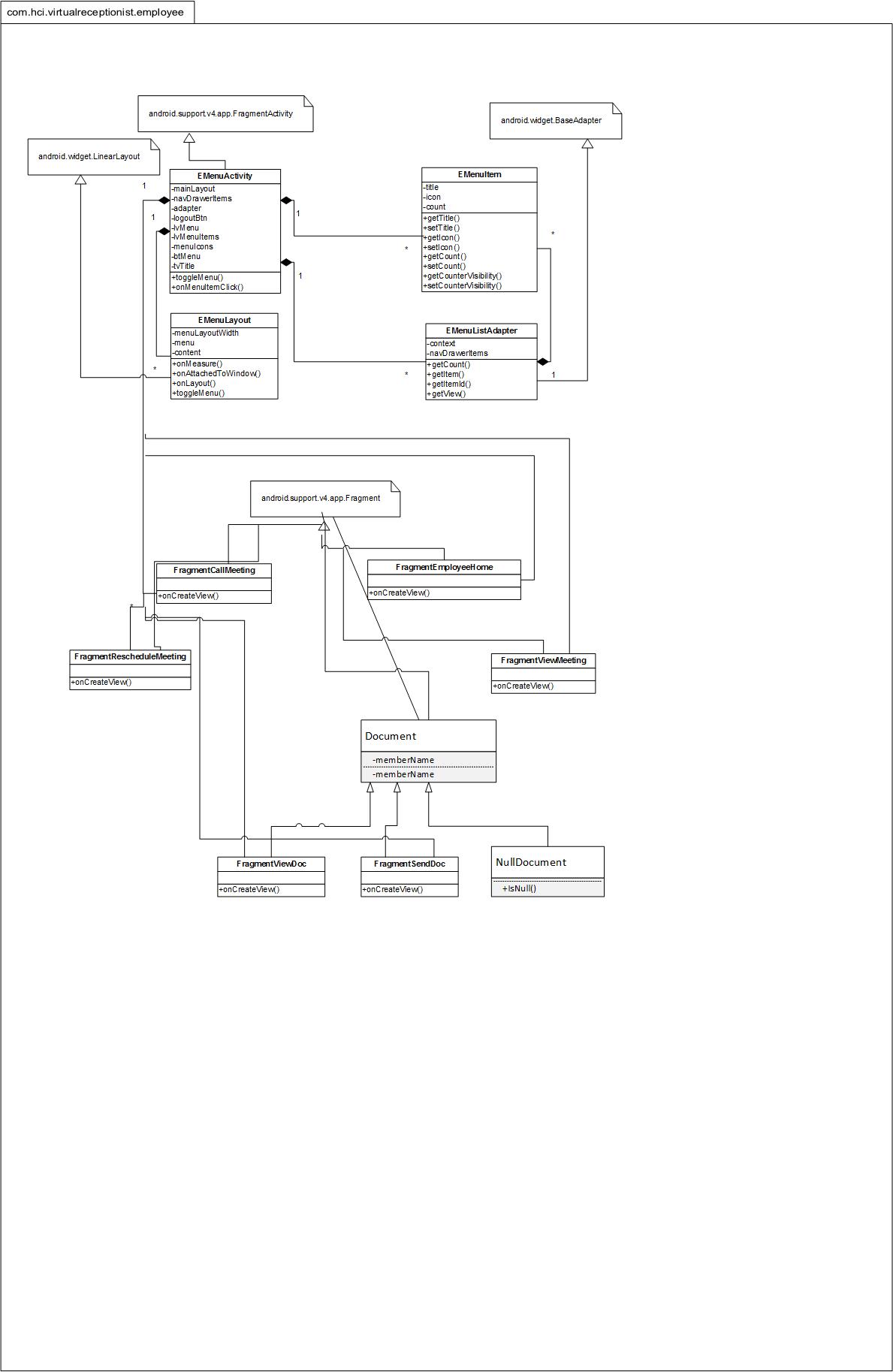


# After Refactoring (Changed Diagrams Only)

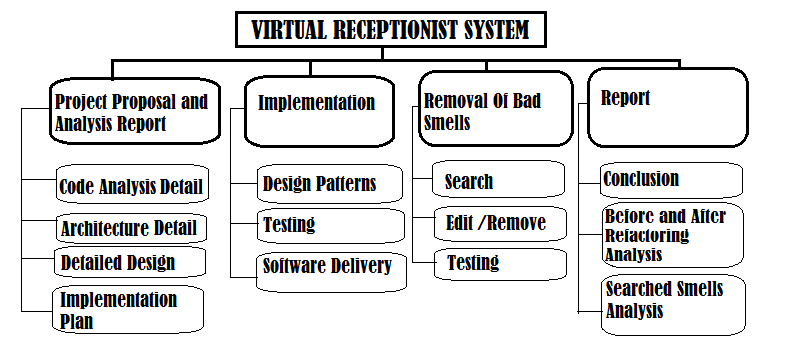
## Main Package



## Employee Package



# Work Breakdown Structure



Members:

K112016 Shoaib Ahmed

K112114 Zohaib Masood

K112116 Fawad Jawaid Malik

K112181 Mohammad Zohair

Design Pattern project has been divided into three phases with 4 main activities i.e. ‘Project Proposal and Analysis Report’, ’Implementation of Patterns into existing Code’, ‘ Removal Of Bad smells aka Refactoring’ , ‘Final Report’. By above WBS diagram you can see all the activities with their sub tasks.

In first of project i.e. ‘**Proposal’**, k112116 and k112016 found out the project to be the project on which we applied design patterns and they analyzed the code where we applied the designed patterns. Then K112114 did some research on the project and analyzed its architecture which we added into our proposal report. Then k112181 wrote down the detailed design of the project and then we all after some discussion wrote down the implementation plan.

In 2nd phase of **‘Implementation’** we applied 8 seven patterns to our project, 4 were applied by the k112116 and k112016 while 4 were applied by k112114 and k112181 through pair programming. Testing was done by every member of the group to ensure the project is working fine.

In 3rd i.e. ’**Refactoring and Final Report**’, bad smells that were present was found by k112114 and k112116, we identified all the bad smells which were making our code looks unprofessional then k112016 and k112181 edit them with the techniques which were taught us in class room. And finally the report in which k112114 made WBS of our project. Class diagram before refactoring by k112016 and after refactoring by k112181 while bad smells, refactoring techniques and conclusion by k112116.

# Concluding Remarks

Our project “Virtual Receptionist” is an android application, due to which, it includes a lot of code which is just used to communicate between GUI and the core of the application. This code was also reused at many places in the project.

Before refraction, it was hard to understand why a block of code was written. The programmer as well as a modifier would have been confused by looking at a bulk of code, just written to carry out tasks, without catering the fact that it would be read by someone. By applying Design Patterns to the project our code has become more managed and efficient. We have also found some of the Bad Smells and refactored it using the techniques available which makes our somehow perfect