ACTIVITY **2.1.1**

Usability Testing

INTRODUCTION

Apps are everywhere. Some apps people return to again and again, while others fall into disuse. In part, an app's fate is tied directly to the purpose that it serves. However, any developer that creates an app without carefully planning for usability and appearance, as well as functionality, will find that their app gets replaced by a competitor before long.

In the next three lessons, you will work on the College App. This work will give you the skills you need to create and thoughtfully present your own app at the end of the unit.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Tools with which to create UML class diagrams

RESOURCES

College App Problem Statement
Resources available online

College App UML
Resources available online

Procedure

Part I: The Problem Statement

The first step in attempting to solve any problem is to understand the problem thoroughly. In Part I, you will redefine and clarify the problem statement.

1. Read **College App Problem Statement**. Record any additional data and/or functionality that you think the College App would benefit from.

2. As directed by your instructor, form groups and discuss the problem statement, as well as your thoughts for improving it.

3. As directed by your instructor, combine your ideas into one master problem statement accessible to everyone in the class.

4. Each group will now be assigned a user **persona** from the list below. Within your group, develop a back story for your persona as directed by your instructor.

   a. Music School applicant

   b. Art School applicant

   c. Transfer student

   d. An applicant who prides herself on her community service work

   e. Trade School applicant

   f. An applicant who does well on standardized tests

   g. A student athlete hoping for an athletic scholarship

   h. Any other user personas your class thinks would be relevant

persona
Fictional person, described in great detail; you design your app with this person in mind as the end user and all design decisions are made to address this person's use cases.

5. As a group, evaluate the master problem statement to determine whether an app designed to solve this problem would address the unique needs of your persona.

6. Under your teacher's direction, discuss your findings from the previous step as a class. Record any additions to the problem statement in the master problem statement.

Part II: Testing for Usability

Usability means how easy or difficult it is for users to interact with your app. In this part, you will examine the usability of the College App that you are planning to build.

7 Your teacher will demonstrate a finished College App. As the demo runs, answer the following questions. If you need to see a particular part of the app run a few times, just ask your teacher.

a. When a user interacts with the app, is it obvious what has happened?

b. Can you tell what kind of data is required?

c. If you make a mistake entering data or navigating within the app, how easy is it to correct that mistake?

d. Based on your use of other apps, how easy was this one to learn?

e. Could any mistakes made using the app have been easily prevented with confirmation dialogs or other preventive measures?

f. Did the app require a user to remember things from one screen to the next?

g. Was there any information provided by the app that did not seem to serve a purpose; how “clean” was the interface?

h. If a user encounters an error message, how helpful was it in allowing you to recover from the error?

i. Were there readily available help functions to assist you in using the app?

8 Compare your notes with a partner and then share with the class as directed by your instructor.

9 Examine another app of your choosing for usability by answering the nine questions above.

NOTE

No games. Games are addressed separately in Unit 4.

Part III: Design of the College App

The last part of this activity is to plan out the design of your College App. In addition to planning what will trigger events in the app, you will also need to decide how to store information in the app.

10 **Events** and **event handlers** are a large part of how Android apps are designed. Reopen the College App on your device. As you use the app

event
Anything that occurs in your app to which you might want to respond with code.

event handler
Code that executes when a given event is detected.

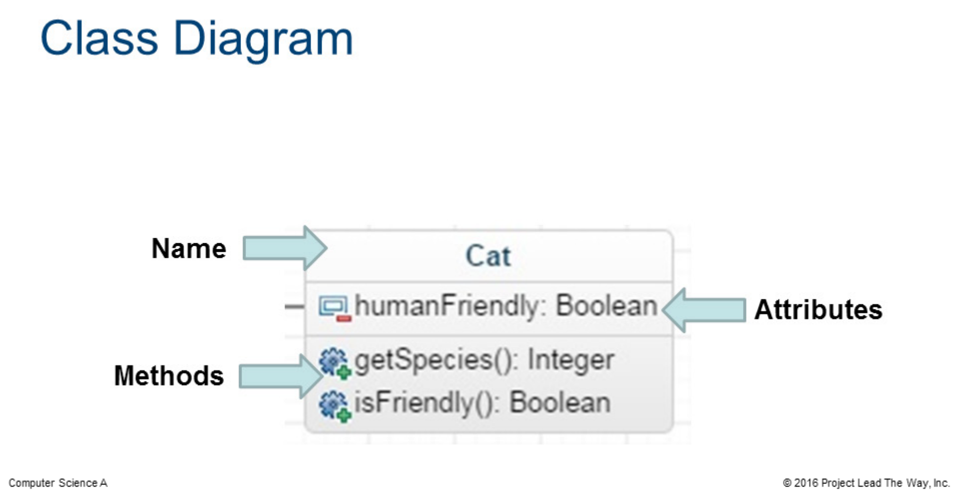
this time, record what events are firing, and what event handlers are being called to execute code when those events occur. You do not need to be concerned with getting the names exactly correct, but try to name them descriptively.

11. In addition to events and event handlers, you will need to create some other classes in the College App. Examining the problem statement and writing down the nouns that you see can be a good starting point to help you think about the classes you will need. Classes should represent whole units in your program and come with their own methods and attributes. For example, the class `Sibling` might have methods `getAge()` and `setAge()` as well as an attribute `age`. Record the classes you think you will need in the College App.

12. As directed by your instructor, discuss your list of classes from the previous step with a partner.

13. Review the UML slideshow and label the **UML** block in the on Slide 4 of the slideshow (also pictured below the slideshow).

UML
Unified Modeling Language: a way of visualizing the relationships between classes in a program.

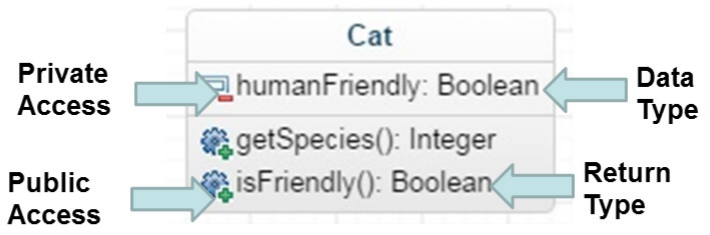

Class Diagram

UML is a tool that is commonly used in industry to plan out the ways in which the different parts (classes) of a program will relate to one another.

A UML diagram may be used to show the methods and attributes that belong to an object as shown here. The name of the class, the attributes belonging to the class, and the methods each have their own regions.

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Class Diagram



Computer Science A © 2016 Project Lead The Way, Inc.

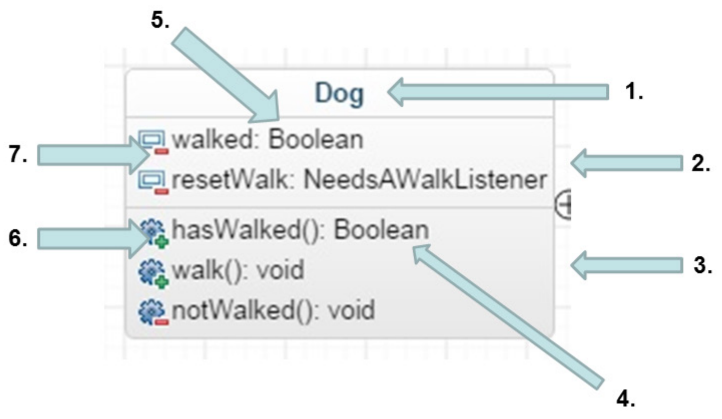
humanFriendly is a member of the Cat class of type Boolean. The variable humanFriendly is private, indicated by a minus symbol, to maintain encapsulation.

Cat has two accessor methods isFriendly() and getSpecies() both of which are public. isFriendly() might be called by the Scheduler class to determine if it is safe for a volunteer to enter the kennel with the cat. getSpecies will be called to determine whether the animal in a kennel is a cat or dog.

Public access is indicated by a plus sign in UML.

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Class Diagram: Test Your Knowledge

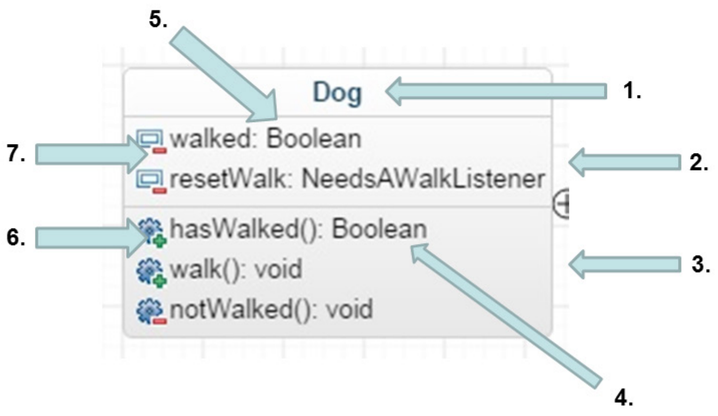


Computer Science A © 2016 Project Lead The Way, Inc.

1. What is the name of this class?
2. What attributes belong to this class?
3. What methods belong to this class?
4. What is the return type of hasWalked?
5. What is the access modifier for walked?
6. What is the access modifier for hasWalked?
7. What is the type of walked?

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

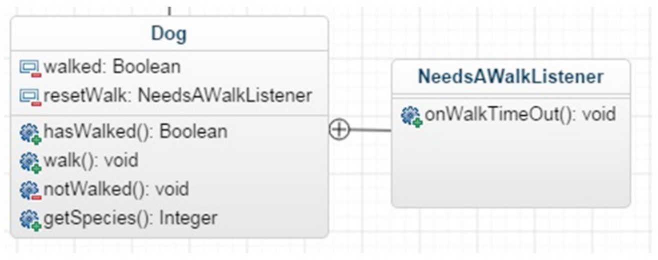
Class Diagram: Test Your Knowledge



1. What is the name of this class? Dog
2. What attributes belong to this class? walked and resetWalk
3. What methods belong to this class? hasWalked(), walk(), and notWalked()
4. What is the return type of hasWalked? Boolean
5. What is the access modifier for walked? private
6. What is the access modifier for hasWalked? public
7. What is the type of walked? Boolean

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

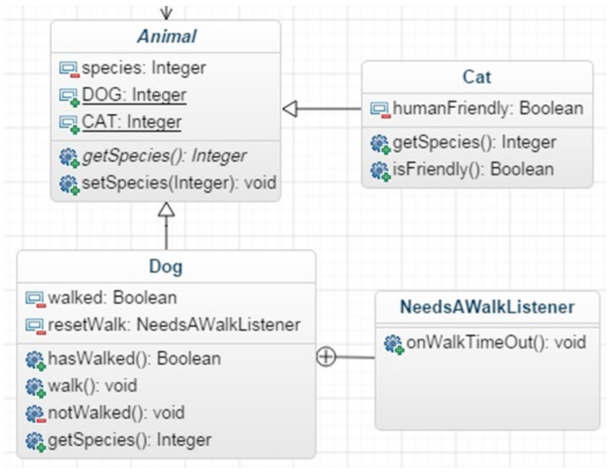
Advanced: Inner Classes



Resetting the value of walked is something that should be done every day, so that the system knows the dog needs to be walked and can assign a Person to do so. That is handled by the NeedsAWalkListener which is an inner class of Dog. An inner class is indicated by the plus sign in the circle next to the class to which the inner class belongs. Inner classes are covered in more detail later in this lesson.

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

“Is a…” Relationship



Both Dog and Cat are Animals. This relationship is shown by the arrows with the closed arrowheads pointing from subclass to superclass. This is known as an inheritance relationship. Because the program might not know what type of Animal is in a given kennel, all kinds of Animals in this implementation have their own getSpecies() methods. These methods can access the DOG and CAT final variables in Animal because the Cat and Dog classes inherit these variables from Animal. Since the implementation of setSpecies(Interger) will not change regardless of species, it only needs to be defined once, in Animal, and then all of the subclasses of Animal, Cat and Dog, automatically gain usage of that method.

You will learn more about inheritance in later activities.

Adaanced:

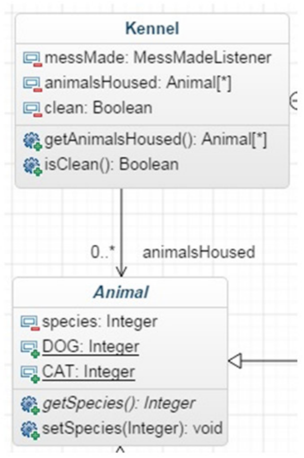
Static methods and attributes are indicated by underlines.

Final class members are indicated only by their naming convention: all capital letters.

Italics indicate that a class or method is abstract. Abstract classes and methods are covered in detail later in the course.

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

“Has a…” Relationship



Computer Science A © 2016 Project Lead The Way, Inc.

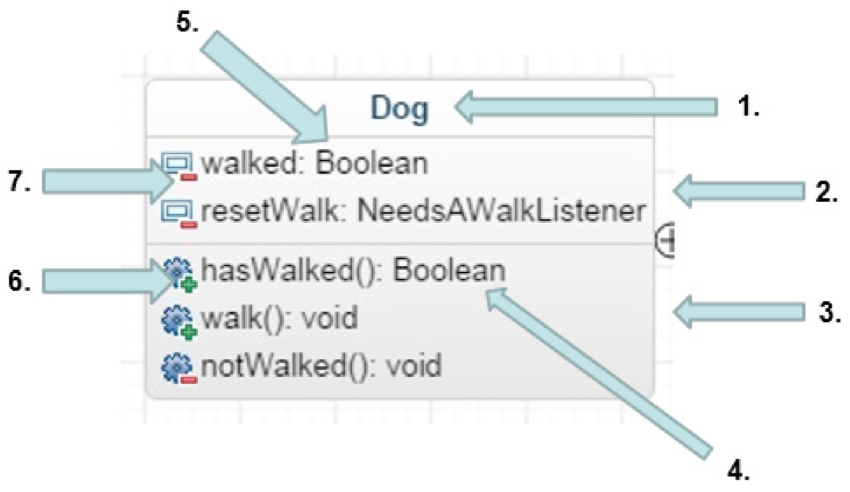
In this diagram, the Kennel class “has an” array, animalsHoused, of type Animal. This is shown by a solid arrow with an open arrowhead. Because the arrow goes in only one direction, this indicates that an Animal object has no knowledge of which Kennel it is associated with.

Advanced:

The 0..* at the arrowhead indicates that a Kennel may have more 0 or more Animal objects associated with it. The name of the member attribute, animalsHoused in this case, accompanies the arrow to show which attribute of Kennel is actively making use of the Animal class.

If a Tricycle class has three instances of a Wheel class, then the 0..* would be replaced with a 3.

For more on UML: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/



14 Create a UML diagram for the College App. Include your classes from the previous step, as well as any methods and attributes belonging to those classes with access modifiers, types, return types, and parameters as appropriate. Use tools specified by your instructor.

15 View **2.1.1 CollegeAppUML** as directed by your instructor. Likely many of the classes you diagrammed in the last step are missing. This is indicative of the lack of functionality you observed in Step 8. Also, there are probably some classes that you didn’t expect. List them.

16 Categorize each of the classes in **College App UML** as being part of the model layer or the presenter layer.

CONCLUSION

1. Reflecting on how the problem statement changed after you added your own contributions, after your group added theirs, after you combined the whole class's thoughts, and after you considered the needs of various usage personas, answer the following:
   a. What do you see as the ideal environment for developing problem statements?
   b. What do you see as the benefits of developing problem statements in an environment like the one you described above?
2. What approach do you take to determine what events and event handlers are likely associated with the following?
   a. A problem statement
   b. An app that has already been created

College App Problem Statement

Introduction

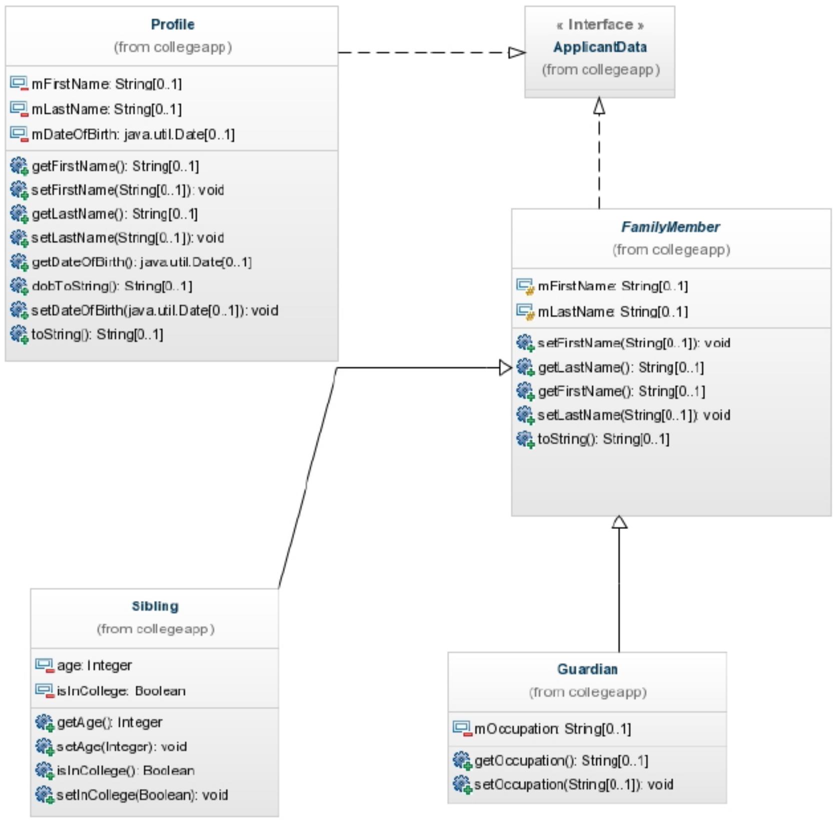
The College App allows an applicant to store and view the data associated with their college application. The data include, but may not be limited to, the applicant's profile, including first name, last name, and date of birth. The applicant should be able to create a list of family members including both guardian and sibling family members. The app should store first and last name of any family member. For guardians, the app should also store occupation. Siblings should have an age. The applicant should also be able to create a list of schools attended; each school should be associated with a start date and an end date. The student should be able to upload either a personal admissions essay or a video recorded to serve the same purpose.

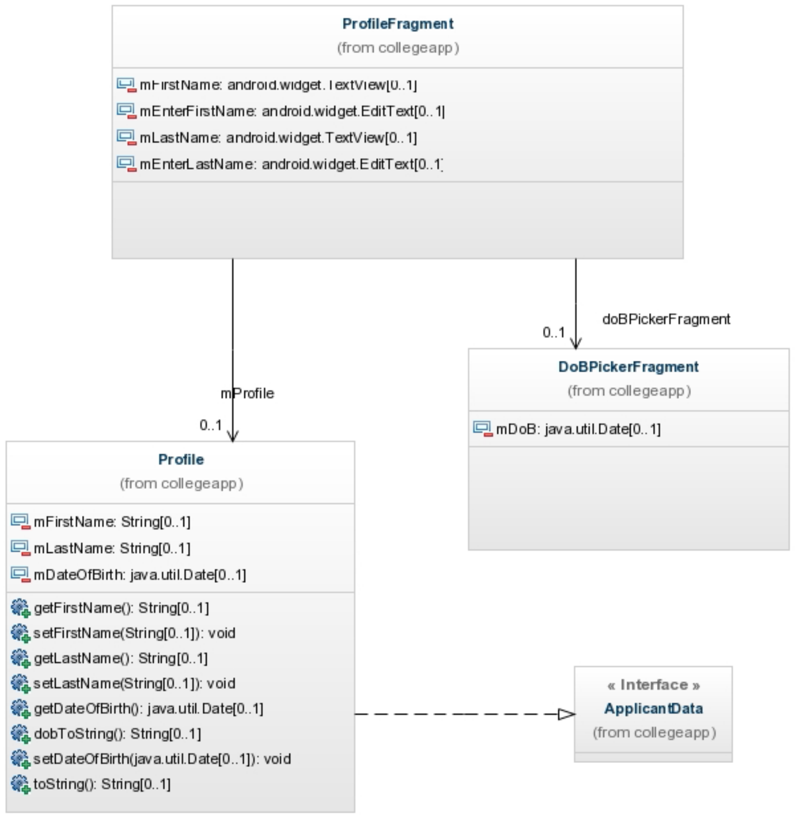
All data should persist if the app is closed or quit, and even if the device powers down.

Context

You have been asked to develop this app by a junior college admissions staff member as a proof of concept. Eventually the hope is that this app would connect to a web server and include secure authentication capabilities for college applicants as well as admissions officials–allowing applicants to enter their data and upload resources from anywhere and allowing officials to review these materials just as easily. For now you are tasked only with the development of features for the applicant that remain local to the device on which they install the app.

College App UML


Profile
(from collegeapp)
mFirstName: String[0..1]
mLastName: String[0..1]
mDateOfBirth: java.util.Date[0..1]
getFirstName(): String[0..1]
setFirstName(String[0..1]): void
getLastName(): String[0..1]
setLastName(String[0..1]): void
getDateOfBirth(): java.util.Date[0..1]
dobToString(): String[0..1]
setDateOfBirth(java.util.Date[0..1]): void
toString(): String[0..1]
« Interface »
ApplicantData
(from collegeapp)
FamilyMember
(from collegeapp)
mFirstName: String[0..1]
mLastName: String[0..1]
setFirstName(String[0..1]): void
getLastName(): String[0..1]
getFirstName(): String[0..1]
setLastName(String[0..1]): void
toString(): String[0..1]
Sibling
(from collegeapp)
age: Integer
isInCollege: Boolean
getAge(): Integer
setAge(Integer): void
isInCollege(): Boolean
setInCollege(Boolean): void
Guardian
(from collegeapp)
mOccupation: String[0..1]
getOccupation(): String[0..1]
setOccupation(String[0..1]): void


ProfileFragment
(from collegeapp)
mFirstName: android.widget.TextView[0..1]
mEnterFirstName: android.widget.EditText[0..1]
mLastName: android.widget.TextView[0..1]
mEnterLastName: android.widget.EditText[0..1]
doBPickerFragment
0..1
DoBPickerFragment
(from collegeapp)
mDoB: java.util.Date[0..1]
mProfile
0..1
Profile
(from collegeapp)
mFirstName: String[0..1]
mLastName: String[0..1]
mDateOfBirth: java.util.Date[0..1]
getFirstName(): String[0..1]
setFirstName(String[0..1]): void
getLastName(): String[0..1]
setLastName(String[0..1]): void
getDateOfBirth(): java.util.Date[0..1]
dobToString(): String[0..1]
setDateOfBirth(java.util.Date[0..1]): void
toString(): String[0..1]
« Interface »
ApplicantData
(from collegeapp)