**CIS 162 Project 2**

**A Simple Cell Phone**

# Due Date

* at the start of lab on October 3

# Before Starting the Project

* Read Chapter 3 and sections 4.1 – 4.2
* Read this entire project description before starting
* Track your time spent working on the project

# Learning Objectives

After completing this project you should be able to:

* *write* methods to meet specific requirements
* *write* conditional statements with Boolean expressions
* *explain* the differences between local variables, instance variables (class fields) and method parameters

# Project Summary

You will create a class to simulate basic functionality of a smart phone. The owner ‘pretends’ to do basic tasks like send texts, stream video (e.g. Netflix), stream audio (e.g. Pandora) and print a monthly statement.

The basic phone plan includes unlimited voice, unlimited texts and 2 gigabytes of data (2 GB) for a base fee of $50. An additional data fee of $15 is charged for each partial GB used above the 2.0 GB provided in the basic plan.

The **phone uses data** for audio streaming at the rate of 65 MB per hour. (Note that 1 GB equals one thousand megabytes, i.e. 1,000 MB).

A **fully charged battery** will last for twelve hours of audio. Texting does not consume battery life.

A typical monthly statement appears below:

MyPhone Monthly Statement

Customer: Amanda Jaffe

Number: (616)999-0000

Texts: 789

Data usage: 2.98 (GB)

2GB Plan: $50.00

Additional data fee: $15.00

Universal Usage (3%): $1.95

Administrative Fee: $0.61

Total Charges: $67.56

# Class Design Specifications for MyPhone

The design for a class includes

1. declarations for a list of instance fields (think of these as nouns) and
2. method declarations (think of these as verbs), including parameters, which determine how objects are used.

Define a class, called *MyPhone*, with the following instance fields and methods. Do not create additional methods or make any changes to the following requirements without approval from your instructor.

**Class Fields**

Declare meaningful names with appropriate data types for each of these private instance fields:

* an integer for the *number of texts*
* a double for the amount of *data consumed* (in megabytes)
* a double for the remaining *battery life* (a value between 0.0 and 1.0)
* a String for the customer name (e.g. “Amanda Johnson”)
* a String for the ten-digit phone number (e.g. “6163319999”)
* a final double for the audio usage per minute. For example:

private final double DATA\_PER\_MIN = 65 / 60.0;

* a final double for the maximum minutes of audio usage for a full battery charge. For example:

private final double MAX\_MINUTES = 720.0;

**Constructor (5 pts)**

A *constructor* is a special method with the same name as the class and generally initializes the fields to appropriate starting values. Refer to section 3.7.

* public MyPhone (String name, String num) – this constructor initializes the instance members to zero and sets the customer name and number to the provided parameters. Note, phones start with no battery life.

**Accessor Methods (5 pts)**

An *accessor* method does not modify class fields. The names for these methods, which simply return the current value of a field, often begin with the prefix ‘get’. Refer to section 3.6.

* public int getNumTexts () - returns the total number of texts sent or received this month.
* public double getBatteryLife () - returns the remaining amount of battery life, should be in the range 0.0 – 1.0.
* public double getDataUsage () - returns the amount of data used so far this month in megabytes (MB).

**Mutator Methods (40 pts)**

A mutator method performs tasks that may modify class fields. Methods which simply set a field with the parameter value often begin with the prefix ‘set’. Refer to section 3.6.

* public void setName (String n) – sets the customer name.
* public void setPhoneNumber (String n) – sets the phone number. The String should contain a ten-digit sequence of digits (e.g. “1234567890”).
* public void chargeBattery(int mins) – updates the battery life to reflect a charge for *mins* additional minutes. A full charge is completed in two hours (120 minutes). Therefore, a 50% charge is completed proportionately in sixty minutes. Charging the battery longer than two hours has no additional effect, battery life should never exceed 1.0. Use JOptionPane to display the updated battery life as a percentage. For example, “Battery Life: 89%”.
* public void streamAudio(int mins) – calculates the amount of data consumed and battery power used to stream the specified minutes of audio. Sixty minutes of audio consumes 65 MB of data. A fully charged battery will last for twelve hours of audio streaming. Update the data and battery life as appropriate. Note, the phone could run out of power before streaming all of the requested audio. In this case, battery life will be 0.0 and only a portion of the requested audio will be streamed. If so, use JOptionPane to display a warning.
* public void sendText (String text) – increments the text counter. Use JOptionPane to display the text.
* public void readText() – uses JOptionPane to display a short text message of your choice. For example, “what are you up to?” For now, the method will always display the same text message.
* public void printStatement() – prints a monthly statement using System.out.print statements. Refer to the sample output for correct format. Your statement should be similar. Pay close attention to text alignment and currency format. Use a NumberFormat object to correctly display monetary values (describe later).

**Private Helper Methods (10 pts)**

Designated as *private*, a helper method is designed to be used by other methods within the class. Good practice is to make methods private unless they need to be public. Refer to section 3.6. Several of the following methods are invoked from within the printStatement method and is an effective way to keep it shorter than it otherwise would be.

* private void startNewMonth() – resets the appropriate instance members to zero. But which ones?
* private double calcAdditionalDataFee() – returns the additional data fee which is $15 for each GB above the basic 2GB, where data usage is rounded up. For example, $15 would be the additional fee for 2.5 GB, $45 would be the additional fee for 4.1 GB, and there would be no additional fee for 1.99GB.
* private double calcUsageCharge() – returns 3% of the phone service cost, where the phone service cost is $50 plus additional data fees.
* private double calcTotalFee() – returns the total monthly fee, which is the sum of the phone service cost, the usage charge, and the administrative fee ($0.61).

private String fmtPhoneNumber() – converts the ten digit String to a standard phone format (e.g. 1234567890 -> (123) 456-7890). You must insert the parentheses and dash into the correct locations. As an example to be modified, the following would create a new string 12345-67890 from the given string 1234567890.

String str = s.substring(0,5) + “-“ + s.substring(5);

**Preventing User Error (10 pts)**

Good programming practice is to avoid, or at least minimize, the effect of user errors. Modify each of the following methods to ignore negative minutes. Battery life and data usage will not change if a negative number is provided as a parameter:

* public void chargeBattery(int mins)
* public void streamAudio(int mins)

Modify the following method.

* public void setPhoneNumber(String n) – sets the phone number to n, but only if the String length is exactly ten. Otherwise, set the number to “9999999999”. Note, this method can be invoked from the constructor to set the initial phone number.

**Personalize Your Solution (10 pts)**

Modify the following method.

* public void readText()- display a random message (from five options). Use the Random class to generate a random number 0 - 4 and then use a switch statement to determine which message to display (section 4.5). Choose your own messages and have some fun :-)

Hint: int choice = rand.nextInt(5);

**Coding Style (10 pts)**

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU [Java Style Guide](http://www.cis.gvsu.edu/java-coding-style-guide/).

**JOptionPane for Output**

System.out.print() is the traditional way to display information in the terminal window. However, JOptionPane is a a more modern way and the information is displayed within a pop up window.

To use JOptionPane, you need to include the compiler directive immediately before (at the top) the class definition.

import javax.swing.JOptionPane;

The following statement will display a message in a pop up window. The first parameter, null, is necessary. Similar to 0, *null* means empty or a reference to nothing in Java.

JOptionPane.showMessageDialog(null, “message goes here”);

**Formatting Numbers**

Pattern your code after the following to format a string with a currency format or with a decimal format within the printStatement method.

# NumberFormat fmt = NumberFormat.getCurrencyInstance();

System.out.println("Cost: " + fmt.format(2.33));

# DecimalFormat fmt2 = new DecimalFormat("#.##");

System.out.println("Quantity: "+ fmt2.format(2.333333));

Will Display

Cost: $2.33

Quantity: 2.33

# Software Testing (10 pts)

Software developers must carefully test their solution. BlueJ allows you to instantiate objects and then invoke individual methods to manipulate the fields (instance variables) within the object. You can call every method with a variety of parameters and compare actual results with expected results. However, this gets tedious and cannot be automated.

## Testing Your Class using the main() method

Another approach is to write a main method that instantiates an object and then calls its various class methods in a carefully designed sequence. Refer to section 4.11.

For this project, write a main method in a new class called MyPhoneTest that instantiates at least two phones for different customer names and invokes each of the methods with a variety of parameter values to test each method. Provide multiple *print* statements and *if* statements to test each method along with error messages as needed. It takes careful consideration to anticipate and test every possibility.

An incomplete example is provided below. Your test method should be much longer.

public static void main(String args[]){

MyPhone mine = new MyPhone("Amanda Jaffe", "1234567890");

MyPhone yours = new MyPhone("Henry Hall", "5555555555");

mine.chargeBattery(120);

mine.streamAudio(360);

mine.chargeBattery(30);

mine.streamAudio(530);

mine.streamAudio(200);

mine.sendText("message");

mine.printStatement();

yours.printStatement();

}

The sample main method will display the following output to the terminal window, plus pop-up windows for 1) “battery life 100%”, 2) “Battery life 75%”, and 3) “Phone needs to be charged”.

myPhone Monthly Statement

Customer: Amanda Jaffe

Number: (123)456-7890

texts: 1

data usage: 0.97 (GB)

2GB Plan $50.00

Additional data fee: $0.00

Universal Usage (3%): $1.50

Administrative Fee $0.61

Total Charges: $52.11

# Sample Results #2

public static void main(String args[]){

MyPhone yours = new MyPhone("Henry Hall", "5555555555");

yours.chargeBattery(120);

yours.streamAudio(700);

yours.chargeBattery(120);

yours.streamAudio(700);

yours.chargeBattery(120);

yours.streamAudio(700);

yours.printStatement();

}

The main() method above will display the following output to the terminal window, plus pop-up windows for 1) “Battery life 100%”, 2) “Battery life 100%”, and 3) “Battery life 100%”.

myPhone Monthly Statement

Customer: Henry Hall

Number: (555)555-5555

texts: 0

data usage: 2.28 (GB)

2GB Plan $50.00

Additional data fee: $15.00

Universal Usage (3%): $1.95

Administrative Fee $0.61

Total Charges: $67.56

# Sample Results #3

public static void main(String args[]){

MyPhone yours = new MyPhone("Henry Hall", "5555555555");

for(int i=1; i <= 6; i++){

yours.chargeBattery(120);

yours.streamAudio(700);

}

for(int j=1; j <=2500; j++){

yours.sendText("CIS 161 is awesome!");

}

yours.printStatement();

}

The main() method above will display the following output to the terminal window, plus six pop-up windows with “Battery life 100%”.

myPhone Monthly Statement

Customer: Henry Hall

Number: (555)555-5555

texts: 2500

data usage: 4.55 (GB)

2GB Plan $50.00

Additional data fee: $45.00

Universal Usage (3%): $2.85

Administrative Fee $0.61

Total Charges: $98.46

# Late Policy

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

* The first 24 hours (-20 pts)
* Each subsequent weekday is an additional -10 pts
* Weekends and university holidays are free days.

# Turn In

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

* Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project. (-5 pts if missing)
* Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/academic-honesty/).
* Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, “I spent 7 hours on this project from September 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document.”
* Sample Output – a printout of the BlueJ Terminal window after running the main method that shows a variety of the printed messages. You can copy and paste into the Word document that contains your cover page.
* Source code - a printout of your elegant source code printed from BlueJ with line numbers (with your name as author).
* Demo – be prepared to demo your project on a lab computer or your laptop. I will ask you to perform a variety of tasks using BlueJ.