1 Submission Instructions

Note that for each homework assignment, only some of the exercises will be graded. A list of the exercises that will be graded can be found in the Module 2: HW1 submission page in Canvas. This does not mean that the ungraded exercise are unimportant or that you will not be tested on the material in the exercise. They are, and you will. There are two primary reasons we grade only some of the exercises: first, this course has a large enrollment so there are many assignments to be graded; second, in the accelerated time frame for online courses, we want to return graded homework exercises as quickly as we can. Despite not grading some exercises, if you wish to perform as well as you can in this course, we strongly recommend that you complete all of the exercises.

Many of your solutions will be typed in a PDF document (the penalty for submitting a document in any format other than PDF will be 10% or -2.5 pts if the grader can open it; if the file cannot be opened, scores of 0 will be assigned to the solutions that are to be included in the PDF).

To start, create a document using your favorite word processor and type your exercise solutions—except for those exercises where you are asked to submit your solution in a separate file (more later). At the top of the document please include your name, your ASURITE ID (the username you use to login to MyASU and Canvas), your email address, and the homework assignment number, e.g. Homework 1.

For **short-answer** and **description** exercises (e.g., see Exs. 3.2, 4.1, 4.2, 5.1–5.8), please neatly type your solution in your document.

Some exercises may ask you to submit an **image** (e.g., see Exs. 7.1–7.2). Hand-drawn images are unacceptable and will result in zero points being assigned for an exercise (the reason being that hand-drawn images are often illegible causing students to lose points). For images, use a software drawing tool, export the image in a suitable image file format, and then copy-and-paste the image file into your word processing document. Please ensure the image is sufficiently sized so that the text is readable.

Please carefully read the following sections below regarding submission of **Java source code files** (Exs. 3.1, 3.4, 3.5, 4.3). We will be using an automated grading script to grade these exercises, so it is extremely important that you follow the assignment instructions—especially in regard to naming—so your submission will not cause the script to fail resulting in point deductions.

If an exercises ask you to write **Java code but does not request you submit your solution** in a separate Java source code file (e.g., see Exs. 3.3, 3.6, 3.7), copy-and-paste your code from your IDE or text editor into your word processing document. Make sure to neatly format your code, i.e., indentation.

When you are done, convert the word processing document into **Adobe PDF format** and name the PDF file *asuriteid-h01.pdf* where *asuriteid* is your ASURITE user id (for example, if your ASURITE user id is *jsmith6* then your file would be named *jsmith6-h01.pdf*).

Next, create an empty folder named *asuriteid-h01* where *asuriteid* is your ASURITE user id and and copy *asuriteid-h01.pdf* to that folder. Copy the requested **zip archives** containing Java source code files for Exs. 4.2–4.5 and 5.1–5.3 into this folder. (Note: Java source code files are the files with a *.java* file name extension; when you create the zip archives for these two groups of exercises, do not include the *.class* files in your zip archives as we do not need those.)

Next, compress the *asuriteid-h01* folder creating a **zip archive** file named *asuriteid-h01.zip*. Upload *asuriteid-h01.zip* to Canvas on the Module 2: HW1 submission page by the assignment deadline. Please see the *Course Schedule* section on the *Syllabus* page in Canvas for the deadline. Consult the Syllabus for the late and academic integrity policies.

1.1 Submitting Java Source Code Files Containing Methods. For Exs. 3.1, 3.4, 3.5—which ask you to submit Java files containing just methods (not a runnable program) that solve the problem—we want you to encapsulate the requested method within a class (i.e., in a *.java* file) named as requested in the exercise. Remember that Java is case-sensitive so your filenames, method names, variable names, etc. must be named exactly as requested in the homework document. Failure to do this will cause you to lose points on an exercise. For example, here is what we want you to submit for Ex. 3.1, where you will complete the code in *arrayListInit*() in a class named *H01_31* to solve the problem:

```
// CLASS:  H01_31
// AUTHOR: your name, your ASURITEID username, your email address
import java.util.ArrayList;  // Import any required classes so your code will build

public class H01_31 {  // Remember that class name and filename have to be the same. Case matters.

    public H01_31() {  // Provide a default constructor. Some exercises may require other ctors.
    }

    // This is the method you are asked to write for Ex. 3.1. Name it exactly as requested.
    public ArrayList<Integer> arrayListInit() {
        // 1. Declare and instantiate an ArrayList<Integer> object named list.
        // 2. Write single for/while loops or a pair of for/while loops to fill list with the specified values.
        // 3. Return list.
    }
}
```

The CLASS: and AUTHOR: header comment block lines must be included. All other comment lines are optional but we strongly encourage you to comment your code. In some situations, well-written comments may help us assign partial credit. Note that the instructor indents using 4 spaces. It does not matter to us how many spaces you indent but please configure your editor to insert *spaces* and not hard tabs when you hit the Tab key. Be sure for each exercise that you import the required classes so that your class will build. It is preferable to import just the one class, e.g., `java.util.ArrayList` rather than writing `import java.util.*;` Given your standalone class, we will write a driver routine that instantiates an object of your class and then call *arrayListInit*() on that object to test your solution. Do not provide a *main*() method or driver routine in your class: for testing on your end, write your own *main*() method and drivers in a class separate from *H01_31*. **For these exercises, you do not need to copy-and-paste your code into the word processing document, but the .java files must be included in your zip archive.**

1.2 Submitting Java Source Code Files for Complete Programs. For exercises which ask you to write complete programs (see Ex. 4.3) your program must contain a *main*() method and any requested drivers. In general, unless the exercise specifically asks you to write class (static) methods, all of the methods in your class(es) except for *main*() must be instance methods. Shown below is the required template for the main class, i.e., the class containing the *main*() method, of a complete program. **For these exercises, you do not need to copy-and-paste your code into the word processing document, but include the .java files in your zip archive.**

```
// CLASS:  MainClassName
// AUTHOR: your name, your ASURITE ID username, your email address
import required stuff ...;  // Do not use * notation

public class MainClassName {

    // Declare instance data as required...

    public static void main(String[] pArgs) { // I often preface parameter variable names with a p
        new MainClassName().run(); // Instantiate an object of this class and then call run() on the object
    }

    public MainClassName() { // Provide a default constructor.
    }

    // Provide other ctors as required...
```

```
    // run() is where the action starts taking place. I like to say that when we leave main() and reach run(), we
    // leave "static land" and enter "object land." Static (class) methods have their uses but they should rarely
    // be used. Note: run() is private because it is intended to be only called from main().
    private void run() {
    }

    // Provide other instance or class methods as required...
}
```

2 Learning Objectives

1. Use the *Integer* and *Double* wrapper classes.
2. Declare and use *ArrayList<E>* class objects.
3. Write code to read from, and write to, text files.
4. Understand exception processing and write an exception handler for I/O exceptions.
5. Write Java classes and instantiate objects of those classes.
6. Read UML class diagrams and convert the diagram into Java classes.
7. Identify and implement dependency, aggregation, inheritance, and composition relationships.

3 ArrayLists

3.1 Learning Objective: To declare, create, and initialize an *ArrayList<Integer>* list.

Instructions: See the instructions in §1.1 for what to submit for grading. This is not a complete program. Name your class *H01_31* and save it in a file named *H01_31.java*. When you are done, copy *H01_31.java* to your *asuriteid-h01* folder, i.e., to the same folder as the PDF.

Problem: Write a public instance method named *ArrayList<Integer> arrayListInit*() that creates an *ArrayList<Integer>* object named *list* and fills *list* with the numbers shown below (using one or a pair of **for** or **while** loops, i.e., if you just call *add*() to add the numbers to *list*, your solution will be marked as incorrect). *list* shall be returned from *arrayListInit*(). Note that your method shall not output the contents of the list, just return it.

```
0  1  2  3  4  0  1  2  3  4
```

Testing: We will be testing your method using our driver routine. For testing on your end, write your own driver routine in a class different than *H01_31*.

3.2 Learning Objective: To use the *ArrayList<E>* class *get*() and *set*() methods.

Problem: Consider the *ArrayList<Integer>* object named *list* containing these *Integers*:

list = { 1, 2, 3, 4, 5, 4, 3, 2, 1, 0 }

What are the contents of *list* after this loop completes?

```
for (int i = 1; i < 10; ++i) {
    list.set(i, list.get(i) + list.get(i-1));
}
```

3.3 Learning Objective: To use an enhanced for loop to iterate over the elements of an *ArrayList<Integer>* list.

Problem: Write an enhanced for loop that counts how many integers in an *ArrayList<Integer>* object named *list* are negative. Print the count after the loop terminates. You may assume that the *list* reference variable is non-null but the list itself may be empty.

3.4 Learning Objective: To compute a running sum of the elements of an *ArrayList<Integer>* list.

Instructions: See the instructions in §1.1 for what to submit for grading. This is not a complete program. Name your class *H01_34* and save it in a file named *H01_34.java*. When you are done, copy *H01_34.java* to your *asuriteid-h01* folder, i.e., to the same folder as the PDF.

Problem: See the instructions in §1.1 for what to submit for grading. This is not a complete program. Name your Java source code file *H01_34.java*. Write a public instance method named *Integer arrayListSum(ArrayList <Integer> pList)*. The method shall return the sum of the elements of *pList* as an *Integer*. Note that the sum of an empty list is 0. If the *pList* reference variable is `null`, then return 0 as well.

Testing: We will be testing your method using our driver routine. For testing on your end, write your own driver routine in a class different than *H01_34*. Make sure to test the cases where list is empty and where the *pList* parameter is `null`.

3.5 Learning Objective: To declare, create, and initialize an *ArrayList<Integer>* list.

Instructions: See the instructions in §1.1 for what to submit for grading. This is not a complete program. Name your class *H01_35* and save it in a file named *H01_35.java*. When you are done, copy *H01_35.java* to your *asuriteid-h01* folder, i.e., to the same folder as the PDF.

Problem: Write a public instance method named named *ArrayList<Integer> arrayListCreate(int pLen, int pInit Value)*. The method shall use a while or for loop to create a new *ArrayList<Integer>* object which has exactly *pLen* elements, *pLen* ≥ 0. Each element shall be initialized to *pInitValue*. The method shall return the *ArrayList*.

Testing: We will be testing your method using our driver routine. For testing on your end, write your own driver routine in a class different than *H01_35*. Make sure to test the case where *pLen* is 0.

3.6 Learning Objective: To insert a *String* into a *ArrayList<String>* list and maintain the sort order of the list.

Problem: Write a void method named *insertName*() that has two input parameters: (1) *pList* which is an object of *ArrayList <String>* where each element of *pList* is a person's name; and (2) a *String* object named *pName*. Assume the names in *pList* are sorted into ascending order. The method shall insert *pName* into *pList* such that the sort order is maintained. The *pList* parameter variable will be non-null but the list itself may be empty.

Algorithm: There are at least two algorithms one could use to solve this problem. The first—and the method you shall not use—is to add pName to the end of pList and then sort pList. The second—and this is the algorithm I want you to use—is to write a ***for*** *or* ***while*** *loop that iterates over the Strings in pList (starting at index 0) until the index is reached where pName must be inserted so pList remains sorted. We will study algorithmic complexity and Big O notation later in the course, so what I say here may not make sense, but the first algorithm has time complexity* $\Theta(n \lg n)$ *and the second is O(n). This means the first algorithm is slower than the second and this is the reason I want you to use the second method.*

Testing: Make sure to test the cases where *pList* is empty, where the new element is added at the beginning of the list, in the middle of the list, and at the end of the list.

3.7 Learning Objective: To remove elements from an *ArrayList<Integer>* list.

Problem: Write a void method named *arrayListRemove*() which has two parameters: *pList* is an object of the *ArrayList <Integer>* class and *pValue* is an int. The method shall remove all elements from *pList* that are equal to *pValue*. The *pList* parameter variable will be non-null but the list itself may be empty.

Testing: Make sure to test the cases where *pList* is empty, where no elements are removed, where all elements are removed, where the only element that is removed is at the beginning of the list, in the middle of the list, and at the end of the list.

4 Text File Input/Output

4.1 Learning Objective: To understand how files are opened.

Problem: Explain what happens if you try to open a file that does not exist for reading.

4.2 Learning Objective: To understand how files are opened.

Problem: Explain what happens if you try to open a file that does not exist for writing.

4.3 Learning Objective: To read and write text files.

Instructions: See the instructions in §1.2 for what to submit for grading. This is complete program with one Java source code file named *H01_43.java* (your main class is named *H01_43*). When you are done, copy *H01_43.java* to your *asuriteid-h01* folder, i.e., to the same folder as the PDF.

Problem: Write a program that prompts the user for the name of a Java source code file (you may assume the file contains Java source code and has a *.java* filename extension; we will not test your program on non-Java source code files). The program shall read the source code file and output the contents to a new file named the same as the input file, but with a *.txt* file name extension, e.g., if the input file is *foo.java* then the output file shall be named *foo.java.txt*. Each line of the input file shall be numbered with a line number (formatted as shown below) in the output file. For example, if the user enters *H01_43.java* as the name of the input file and *H01_43.java* contains:

```
//*************************************************************
// CLASS: H01_43
//*************************************************************
public class H01_43 {
    public static void main(String[] pArgs) {
    }
    private run() {
    }
}
```

then the contents of the output file *H01_43.java.txt* would be:

```
[001] //*************************************************************
[002] // CLASS: H01_43
[003] //*************************************************************
[004] public class H01_43 {
[005]     public static void main(String[] pArgs) {
[006]     }
[007]     private run() {
[008]     }
[009] }
```

You may assume that the file being read exists.

Hint: To print an integer as shown above in a field of width 3 with leading 0's use the Java API *PrintWriter.printf*() method with a format specifier of `%03d`. The `%` symbol tells the compiler this is a format specifier, 3 is the field width for the integer line number, the 0 preceding 3 tells the compiler to output leading 0's rather than leading spaces if the line number does not take up the entire field width, and `d` tells the compiler that we are printing an integer).

Testing: Make sure to test the case where the input file exists but is empty.

5 Exceptions and Exception Handling

5.1 Learning Objective: To demonstrate knowledge of throwing and catching exceptions.

Problem: Explain the difference between throwing an exception and catching an exception, i.e., explain what happens when an exception is thrown and when one is caught?

5.2 Learning Objective: To demonstrate knowledge of checked and unchecked exceptions.

Problem: Explain what a checked exception is. Give one example.

5.3 Learning Objective: To demonstrate knowledge of checked and unchecked exceptions.

Instructions: Include your answer in *asuriteid-h01.pdf*.

Problem: Explain what an unchecked exception is. Give one example.

5.4 Learning Objective: To demonstrate knowledge of exception handling.

Problem: Which type of uncaught exceptions must be declared with the throws reserved word in a method header?

5.5 Learning Objective: To demonstrate knowledge of exception handling.

Instructions: Include your answer in *asuriteid-h01.pdf*.

Problem: Why don't you need to declare that your method might throw an *IndexOutOfBoundsException*?

5.6 Learning Objective: To demonstrate knowledge of exception handling.

Instructions: Include your answer in *asuriteid-h01.pdf*.

Problem: Is the type of the exception object that gets thrown always the same as the exception type declared in the catch clause that catches the exception? If not, why not?

5.7 Learning Objective: To demonstrate knowledge of exception handling.

Problem: What is the purpose of the finally clause? Give an example of how it can be used.

5.8 Learning Objective: To demonstrate knowledge of exception handling in the *Scanner* class.

Problem: Which exceptions can the *next*() and *nextInt*() methods of the *Scanner* class throw? Are they checked exceptions or unchecked exceptions?

6 Objects and Classes

6.1 Learning Objective: To demonstrate knowledge of instance and class methods.

Problem: Explain how an instance method differs from a class method (static method).

6.2 Learning Objective: To demonstrate knowledge of class constructors.

Problem: Explain what happens if we write a class named *C* but do not implement any constructors in *C*.

6.3 Learning Objective: To demonstrate knowledge of instance and class methods.

Instructions: Include your answers in *asuriteid-h01.pdf*.

Problem: (a) In a static method, it is easy to differentiate between calls to instance methods and calls to static methods. How do you tell the method calls apart? (b) Why is it not as easy to distinguish between calls to instance and static methods which are called from an instance method?

6.4 Learning Objective: To demonstrate knowledge of constructors, instantiating objects, and calling methods.

Problem: Explain what happens when this application runs and why.

```
public class C {
    private int x;
    private String s;
    public static void main(String[] pArgs) {
        new C();
    }
    public C() {
        x = s.length();
        System.out.println("x = " + x);
    }
}
```

6.5 Learning Objective: To declare a complete class, declare instance and class data, instance and class methods, constants, and to understand the differences between public and private accessibility modifiers.

Instructions: See the instructions in §1.1 for what to submit for grading. This is not a complete program. Name your class *H01_65* and save it in a file named *H01_65.java*. When you are done, copy *H01_65.java* to your *asuriteid-h01* folder, i.e., to the same folder as the PDF.

Create a public class named *H01_65* and write the class declaration for *H01_65* that declares: (1) a private int instance variable named mX; (2) a private int class variable named mY initialized to 0; (3) a private int class constant named A which is equivalent to 100; (4) a public int class constant named B which is equivalent to 200; (5) public accessor and mutator methods for mX named *getX*() and *setX*(); (6) public accessor and mutator methods for mY named *getY*() and *setY*(); (7) a public constructor that has one int input parameter named pX which calls *setX*() to initialize mX to pX; (8) a public default constructor that calls *H01_65*(int) to initialize mX to -1.

6.6 Learning Objective: To declare an object of a class, calling a constructor.

Instructions: Continuing the previous exercise, create a class named *H01_66* in *H01_66.java*. This is the main class for the program: see §1.2 for the required template. There is an exception to the template for this exercise: omit the *run*() method and in *main*() do not instantiate an object of the class and call *run*() on the object; rather, for Exercises 6.6–6.8, all of the requested statements will be written in *main*().

Problem: Within *main*() write a statement which instantiates a *H01_65* object named *cObj1* by calling the default constructor.

Testing: This program will verify that there are no syntax errors in *H01_65* and that the default constructor and *setX*() are correctly implemented.

6.7 Learning Objective: To declare an object of a class, calling a constructor.

Instructions: Continue the previous exercise by modifying *main*().

Problem: Write a statement in *main*()—below the statement you wrote for Ex. 6.6—that instantiates another *H01_65* object named *cObj2* by calling the second constructor to initialize the mX instance variable to 10.

Testing: This program will verify that there are no syntax errors in *H01_65* and that the second constructor and *setX*() are correctly implemented.

6.8 Learning Objective: To demonstrate knowledge of accessing the public/private data and methods of a class.

Instructions: There are short answer questions. Type your solution in your PDF document.

Problem: Continuing the previous exercise. Within *main*(), are the following statements syntactically legal, i.e., do they compile? If so, describe what happens when the statement is executed. If not, explain why the statement is syntactically illegal. You may write these statements in *main*() to solve this exercise, but do not include these statements in the Java file you submit for grading.

(a) `int a1 = H01_65.mX;`
(b) `int a2 = H01_65.mY;`
(c) `int a3 = H01_65.A;`
(d) `int a4 = H01_65.B;`
(e) `cObj1.H01_65(20);`
(f) `int a5 = cObj1.getX();`
(g) `cObj1.setX(20);`
(h) `cObj2.setX(cObj1.getX());`
(i) `int a6 = H01_65.getX();`
(j) `H01_65.setX(20);`
(k) `int a7 = cObj1.getY();`
(l) `cObj1.setY(20);`
(m) `int a8 = H01_65.getY();`
(n) `H01_65.setY(20);`

6.9 Learning Objective: To demonstrate knowledge of accessing the public/private data and methods of a class.

Problem: Continuing the previous exercise. Suppose we add these two methods to the declaration of class *H01_65*. For each assignment statement, if it is legal (compiles) explain what happens when the statement is executed. For each assignment statement that is illegal (syntax error) explain why the code is illegal. Do not include these methods in the *H01_65.java* file that you submit for grading.

```
public void f() {
    mX = 0;
    mY = 0;
}
```

```
public static void g() {
    mX = 0;
    mY = 0;
}
```

7 Object Oriented Design and UML Class Diagrams

7.1 Learning Objectives: To draw a UML class diagram.

Instructions: See §1 *Submission Instructions* for the instructions regarding images in exercise solutions. This exercise is not graded so you do not need to insert your image into *asruiteid-h01.pdf.*

Problem: Below is the code for a Java class named *C*. Using a UML class diagram drawing tool, draw the UML class diagram that corresponds to this code. There are several free and open-source tools for drawing UML diagrams. Two reasonably simple-to-use and cross-platform (Windows, Mac, Linux) tools are:

Umlet – http://www.umlet.com
Violet – http://alexdp.free.fr/violetumleditor

The instructor has been using Umlet for all of the class diagrams he has drawn for the lecture notes and he highly recommends it as it is very easy to learn how to quickly draw a nice-looking class diagram. Using Umlet, you can export your diagram as an image file by selecting File | Export As... on the main menu. We recommend you export the diagram as an EPS (Encapsulated Postscript) image and then copy-and paste the EPS image into your word processing document (I do not know if it is just Libre Office or if this would happen in MS Office as well, but I find that EPS images look significantly better and are more readable when pasted into a Libre Office document than PNG or JPG images). If EPS does not work, then export the image as a PNG image.

```
public class C {
    public static final int CONST1 = 100;
    private int mX;
    private double mY;
    private String mZ;
    public C() { this(0, 0, ""); }
    public C(int pX, double pY, String pZ) { setX(pX); setY(pY); setZ(pZ); }
```

```
    public int getX() { return mX; }
    public int getY() { return mY; }
    private String getZ() { return mZ; }
    public void setX(int pX) { mX = pX; }
    public void setY(int pY) { mY = pY; }
    private void setZ(int pZ) { mZ = pZ; }
}
```

7.2 Learning Objectives: To draw a UML class diagram with relationships.

Instructions: Shown below is a UML class diagram which shows several classes that are associated in various ways (this diagram was drawn with Umlet and is in the file *class-relation.uxf* in the homework zip archive). Using Umlet or some other UML class diagram tool, modify the image as requested in the Problem statement and insert the resulting image into *asuriteid-h01.pdf*.

Note: Multiplicities do not apply to generalization/inheritance relationships because generalization is always 1 to 1.

Problem: **(a)** Note that in *Course* there is an instance variable *mRoster* which is an instance of *Roster*. When a *Course* object is deleted, the *Roster* instance *mRoster* will also be deleted. Given that, what type of class relationship, if any, exists between *Course* and *Roster*? If there is a relationship, modify the diagram to indicate the relationship and label each end of the relationship with a multiplicity. **(b)** Note that in *Roster* there is an instance variable *mStudents* which is an *ArrayList* of *Student* objects. When a *Roster* object is deleted, *mStudents* is also deleted, but each *Student* object within *mStudents* is not deleted. Given that, what type of class relationship, if any, exists between *Roster* and *Student*. If there is a relationship, modify the diagram to indicate the relationship and label each end of the relationship with a multiplicity. **(c)** What type of class relationship, if any, exists between Student and UndergradStudent? If there is a relationship, modify the diagram to indicate the relationship (note that we do not use multiplicities on superclass/subclass relationships). **(d)** What type of class relationship, if any, exists between Student and GradStudent? If there is a relationship, modify the diagram to indicate the relationship. Should there be a multiplicity applied to this relationship? If you think so, then indicate it. **(e)** What type of class relationship, if any, exists between UndergradStudent and GradStudent? If there is a relationship, modify the diagram to indicate the relationship.