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 Homework Assignment 2 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 2.

For **short-answer** and **description** exercises (e.g., see Exs. 3.2, 3.6, 3.7, 3.8, 3.10, 3.14, and 6.2), please neatly type your solution in your document.

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. 3.6–3.7 and 3.15), 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.

Please carefully read the instructions in Exs. 4.2–4.5 and 5.1–5.3 regarding submission of **Java source code files**. These two groups of exercises ask you to write complete programs. We will be using an automated grading script to grade these exercises, so it is extremely important that you follow the exercise instructions—especially in regard to naming of things such as filenames, classnames, etc.—so your submission will not cause the script to fail resulting in point deductions.

When you are done, convert the word processing document into **Adobe PDF format** and name the PDF file *asuriteid-h02.pdf* where *asuriteid* is your [ASURITE user id](#) (for example, if your ASURITE user id is *jsmith6* then your file would be named *jsmith6-h02.pdf*).

Next, create an empty folder named *asuriteid-h02* where *asuriteid* is your ASURITE user id and and copy *asuriteid-h02.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-h02* folder creating a **zip archive** file named *asuriteid-h02.zip*. Upload *asuriteid-h02.zip* to the Canvas on the *Module 4: Homework Assignment 2 Submission* page by the assignment deadline. Please see the *Course Summary* section on the *Syllabus* page in Canvas for the deadline. Consult the Syllabus for the late and academic integrity policies.

2 Learning Objectives

1. Properly use the accessibility specifiers public, protected, private in the design and implementation of an OO program.
2. Write Java code to override and overload methods.
3. Properly use inheritance in the design and implementation of an OO program.
4. Properly use polymorphism in the design and implementation of an OO program.
5. Know when to declare a class versus an interface. Declare and implement a Java interface.
6. Implement a GUI using the Java Swing library.

3 Objects, Classes, and Inheritance

3.1 Learning Objective: To effectively use accessor and mutator methods.

Problem: Is it required to provide an accessor and/or mutator method for every instance variable of a class? If yes, explain why this is required, and if no, explain why not.

3.2 Learning Objective: Superclass references and subclass objects; Subclass references and superclass objects.

Instructions: Type the solutions to sub-exercises (a)–(d) in *asurite-h02.pdf*.

Problem: Suppose the class *Sub* extends *Sandwich*. Which of the following statements (a)–(d) are legal?

```
Sandwich x = new Sandwich();
Sub y = new Sub();

(a) x = y;
(b) y = x;
(c) Sub y = new Sandwich();
(d) Sandwich x = new Sub();
```

3.3 Learning Objective: To understand the relationship between superclass and subclass attributes.

Problem: True or False? A subclass declaration will generally contain declarations for instance variables that are specific to objects of that subclass, i.e., those instance variables represent attributes that are not part of superclass objects.

3.4 Learning Objective: To understand the relationship between superclass and subclass attributes.

Problem: True or False? A superclass declaration will generally contain declarations for instance variables that are specific to objects of that superclass, i.e., those instance variables represent attributes that are not part of subclass objects.

3.5 Learning Objectives: To understand the accessibility relationships among superclasses and subclasses.

Problem: Consider classes C1, C2, and C3. Answer the following questions. Regarding a class's instance variables, by "directly accessible", we mean that if we are executing, say, c2Method1(), is it syntactically legal to write a statement that accesses, say, instance variable x3 of class C1? For example, in c2Method1(), is --x3; a legal statement? Similarly, regarding a class's instance methods, by "callable", we mean that if we are executing, say, c3Method2(), is it syntactically legal to write a statement c1Method2(); to call that method which is declared in C1?

```
class C1 {                      class C2 extends C1 {           class C3 extends C2 {
   public int x1;                  public int y1;                  public int z1;
   protected int x2;               protected int y2;               protected int z2;
   private int x3;                 private int y3;                 private int z3;
   public void c1Method1() {}      public void c2Method1() {}      public void c3Method1() {}
   protected void c1Method2() {}   protected void c2Method2() {}   protected void c3Method2() {}
   private void c1Method3() {}     private void c2Method3() {}     private void c3Method3() {}
}                               }                               }
```

1. Which instance variables x1, x2, x3 declared in C1 are directly accessible in c1Method1()?
2. Same variables, in c1Method2()?
3. Same variables, in c1Method3()?

4. Which instance variables x1, x2, x3 declared in C1 are directly accessible in c2Method1()?
5. Same variables, in c2Method2()?
6. Same variables, in c2Method3()?

7. Which instance variables x1, x2, x3 declared in C1 are directly accessible in c3Method1()?
8. Same variables, in c3Method2()?
9. Same variables, in c3Method3()?

10. Which instance variables y1, y2, y3 declared in C2 are directly accessible in c1Method1()?
11. In c1Method2()?
12. In c1Method3()?

13. Which instance variables z1, z2, z3 declared in C3 are directly accessible in c1Method1()?
14. Same variables, in c1Method2()?
15. Same variables, in c1Method3()?

16. Which instance methods c1Method1(), c1Method2(), c1Method3() are callable from c2Method1()?
17. Which C1 methods are callable from c2Method2()?
18. Which C1 methods are callable from c2Method3()?

19. Which instance methods c1Method1(), c1Method2(), c1Method3() are callable from c3Method1()?
20. Which C1 methods are callable from c3Method2()?
21. Which C1 methods are callable from c3Method3()?

22. Which instance methods c2Method1(), c2Method2(), c2Method3() are callable from c1Method1()?
23. Which C2 methods are callable from c1Method2()?
24. Which C2 methods are callable from c1Method3()?

25. A C1 object is created as a block of memory. Within that block of memory, how many instance variables exist?
26. A C2 object is created as a block of memory. Within that block of memory, how many instance variables exist?
27. A C3 object is created as a block of memory. Within that block of memory, how many instance variables exist?

3.6 Learning Objective: To effectively write and call overloaded methods.

Instructions: Type the solution in *asurite-h02.pdf*.

Problem: Explain what an overloaded method is and give an example by writing some Java code.

3.7 Learning Objective: To effectively write and call overridden methods.

Instructions: Type the solution in *asurite-h02.pdf*.

Problem: Explain what an overridden method is and give an example by writing some Java code.

3.8 Learning Objective: To recognize when accidental overloading occurs and how to write Java code to prevent it.

Instructions: Type the solution in *asurite-h02.pdf*.

Problem: Explain what accidental overloading is and the preferred Java method for preventing it.

3.9 Learning Objective: To effectively write and call overridden methods.

Problem: If an overridden method in a subclass needs to call the overridden superclass method, how is this accomplished?

3.10 Learning Objective: To effectively write and call overloaded methods.

Instructions: Type the solution in *asurite-h02.pdf*.

Problem: True or False? It is legal to write a method in a class which overloads another method declared in the same class. Explain.

3.11 Learning Objective: To effectively write and call overridden methods.

Problem: True or False? It is legal to write a method in a class which overrides another method declared in the same class. Explain.

3.12 Learning Objective: To effectively write and call overloaded methods.

Problem: True or False? It is legal in a subclass to write a method which overloads a method declared in the superclass. Explain.

3.13 Learning Objective: To effectively write and call overridden methods.

Problem: True or False? It is legal to write a method in a subclass which overrides a method declared in the superclass. Explain.

3.14 *This exercise was eliminated but left here so the remaining exercise numbers would not change.*

3.15 Learning Objective: To effectively write and call overridden methods.

Problem: True or False? It is legal to write a method in a superclass which overrides a method declared in a subclass. Explain.

3.16 Learning Objective: To effective call superclass constructors from a subclass constructor.

Problem: In a subclass constructor, the superclass default constructor is automatically called before the statements of the subclass constructor begin executing. Suppose we wish to call a different superclass constructor (i.e., not the default constructor) from the subclass constructor. Explain how this is accomplished and give a Java example.

3.17 Explain how an abstract class differs in at least one way from a concrete class.

4 Objects, Classes, Polymorphism, and Interfaces

Learning Objectives: The objectives for Exercises 4.1–4.5 are to effectively declare and implement Java interfaces and to properly use polymorphism.

Instructions: You will submit this program for grading. Create an empty folder named *h02_45*. Copy *Amphibian.java*, *Bee.java*, *Cat.java*, *Cricket.java*, *Dog.java*, *Frog.java*, *Insect.java*, *Main.java*, *MakesSound.java*, and *Mammal.java* to *h02_45*. Do not include any *.class* files in *h02_45* as we will build your program for grading. Then compress *h02_45* creating a zip archive named *h02_45.zip*. Include *h02_45.zip* in your *asurite-h02.zip* zip archive that you submit for grading.

4.1 Problem: The Homework 2 zip archive contains a folder *cse205-h02/src/h02_45* containing the source code for the *MakesSound* interface and the *Mammal*, *Cat*, *Dog*, *Insect*, and *Cricket* classes discussed in the Week 3 video lecture for *Interfaces : Section 6*. In this lecture, we discussed an example program that implements an inheritance hierarchy (*Mammal* is the superclass of *Cat* and *Dog*; *Insect* is the superclass of *Cricket*). Which method or methods in this program are called polymorphically?

4.2 See the instructions in Ex. 4.5 for what to submit for grading. Name your Java source code file *Bee.java*. Write the Java code to declare a new class *Bee* which is a subclass of *Insect*. The sound made by a *Bee* is "Buzz".

4.3 See the instructions in Ex. 4.5 for what to submit for grading. Name your Java source code file *Amphibian.java*. Write the Java code to declare a new abstract class *Amphibian* that implements the *MakesSound* interface.

4.4 See the instructions in Ex. 4.5 for what to submit for grading. Name your Java source code file *Frog.java*. Write the Java code to declare a new class *Frog* which is a subclass of *Amphibian*. The sound made by a *Frog* is "Ribbet".

4.5 To complete this exercise, you must first complete Exs. 4.2–4.4. Then, modify the *run*() method of *Main* and add a few *Bee*s and *Frog*s to the *critters* list. Build your program and verify that it works correctly, i.e., that each critter makes the correct sound.

5 GUI Programming

Learning Objectives: To implement a simple GUI using the Java Swing graphics library. To implement functionality to detect and respond to user interface events.

Instructions: You will submit this program for grading. Create an empty folder named *h02_51*. Copy *Main.java* and *View.java* to *h02_51*. Do not include any *.class* files in *h02_51* as we will build your program for grading. Then compress *h02_51* creating a zip archive named *h02_51.zip*. Include *h02_51.zip* in your *asurite-h02.zip* zip archive that you submit for grading.

5.1   The Homework 2 zip archive contains a folder */src/h02_51* containing two source code files named *Main.java* and *View.java*. *Main* is implemented for you. For Exs. 5.1–5.3 you will modify the *View* class to implement a GUI interface for a calculator as shown below. The calculator will not be fully functional, i.e, it will not perform arithmetic; the primary learning objective of these exercises is to learn how to implement a GUI in Java using the Swing library. For this particular exercise, modify *View.java* in the places where the code is not completed (indicated by ??? symbols in *View*) and implement the code as specified by the comments so when the program is run, the calculator GUI shown below will appear.
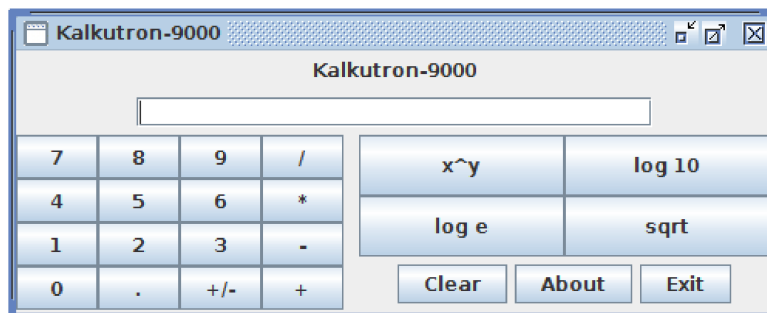
5.2   Complete the code in *actionPerformed*() so when the Exit button is clicked, the application will terminate.

5.3   Complete the code in *actionPerformed*() so when the About button is clicked, the application will display this about dialog where you are to put your own name in the location where *Put your name here* is displayed.
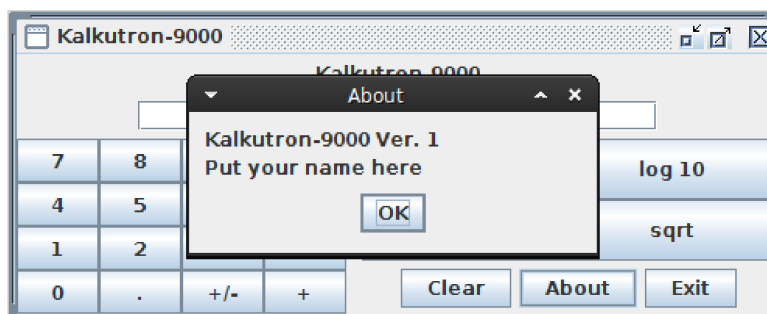
6 Nested Classes

6.1 Learning Objective: To effectively use inner classes.

Problem: Explain what an inner class is.

6.2 Learning Objective: To effectively use local and inner classes.

Instructions: Type the solution in *asurite-h02.pdf*.

Problem: Explain how a local class differs from an inner class.

6.3 Learning Objective: To properly use anonymous, inner, and local classes.

Problem: Explain how an anonymous class differs from an inner and local class.