**Score: \_\_\_\_\_**

**HA1 – Introduction to Computing Abstractions**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

**Introduction:**

In defining abstraction John Guttag says that in a given context relevant information is preserved and irrelevant information can be forgotten. It is also possible, if you are *using* an abstraction, as in the examples from class (TV remote, randint, Dictionary) that you never have to know the irrelevant information, it is completely hidden from you. However, If you are *creating* an abstraction, then you will have to know all of the information in order to design and implement the abstraction. For example, think about the person who created the TV remote or wrote the code for Python’s randint function or Dictionary class. Of course, once these abstractions have been created the irrelevant information is hidden from someone who is using them.

It is not required viewing, but if as you progress through these activities you would like another explanation of abstraction check out the video *Computer Architecture – Abstraction* from Google’s IT Support Professional Certificate course:

* <https://www.youtube.com/watch?v=cjqPMf2gzeQ> (2:18)

🔑 1. For the television remote control example from class identify some of the relevant information is that is retained by the abstraction and describe some of the irrelevant information is that is hidden by the abstraction.

🔑 2. Describe another abstraction from the physical world that you use on a regular basis. Clearly identify the relevant information that you need to use the abstraction and the irrelevant information that is hidden from you.

🏆 3. Give another example of an abstraction that you have used in programming. Provide a snippet of code (maybe from a past assignment in another course) that illustrates the use of the abstraction. Clearly identify what the abstraction is, describe the relevant information that you need to know to use it and the irrelevant information that it hides from you.

**Abstractions are Imperfect:**

Sometimes when we use an abstraction and know only the “relevant” details the behavior of the abstraction can seem amazing or bizarre or downright incorrect. I use quotes here around “relevant” because, as the following examples will show, sometimes some of the “irrelevant” information that is hidden can become very relevant.

In class we looked at an example where thinking simply of an int as an abstraction for an integer caused “strange behavior.” The “relevant” information that we typically use is that an int is an integer number. Based just on that abstraction, if we continually add 1 to the variable it should have simply continued to increase forever and the loop should never end. But we saw that it did end. So somehow repeatedly adding 1 to a positive value eventually made it negative – “bizarre!” The only way to understand that behavior will be to dig deeper. That is, we’ll need to move down the abstraction hierarchy and unhide some of the “irrelevant” details that are hidden by the int abstraction. We will do that in a few days. For now, you’ll look at a few other examples where the hidden or forgotten information is relevant to understanding the behavior of abstractions you’ve been using in your Java programs.

🔑 4. Consider the following Java program that is quite similar to the one we saw in class:

* <https://replit.com/@braughtg/WhileLoop2>
  + Click on “Show files”
  + Look at Main.java

Note: If you have your own Replit account you can use the “Fork repl” button to copy the program to your account where you will be able to edit and run it.

a. Study the program. Then explain in a few sentences what the program would do and why it would do it if the int variable i was a perfect abstraction for an integer.

b. Now run the program. What output does it generate?

c. What must have happened to the value of the variable i in order to see the output your observed in part b? Briefly explain how do you know that.

🔑 5. Consider the following program in the repl.it environment:

* <https://replit.com/@braughtg/Complement>

a. Study the program and then give the output that you would expect it to generate if the int variable i was a perfect abstraction for an integer.

b. Now run the program. What output did it generate?

c. Briefly explain why the output of this program would be surprising to someone thinking that the int variable i is a perfect abstraction for an integer.

🏆 6. Consider the following program in the repl.it environment:

* <https://replit.com/@braughtg/intMultiply>

a. Study the program and then give the output that you would expect it to generate if the int variable i was actually a perfect abstraction for an integer.

b. Now run the program. What output does it generate?

c. Explain as clearly as you can, how the behavior of this program is related to that of the WhileLoop1 program (<https://replit.com/@braughtg/>[WhileLoop1](https://replit.com/@braughtg/WhileLoop1)) that we saw in class.

🏆 7. Consider the following program in the repl.it environment:

* <https://replit.com/@braughtg/doubleMultiply>

Let’s analyze this program.

a. The variable f contains the value 4.35. Given that, what should the value of the variable g be? Include at least one decimal place in your answer (i.e. xxx.0)

b. The value of the variable n is set by type casting the value of g from a double value to an int value. Recall that a type cast from a double to an int simply discards any decimal portion of the value (i.e. it does not round.).

Tip: If you don’t recall exactly how type casting works in Java, you can review the examples of “narrowing” or “explicit” type casts where double values are being converted to int values at the following sites:

* + <https://www.w3schools.com/java/java_type_casting.asp>
  + <https://www.section.io/engineering-education/type-casting-in-java/>

Based on your answer to part a, what value should the variable n have?

c. Run the program. Based on the output you see, what was the actual value of the variable n?

d. Given the output observed in part d, the value of g could not possibly have been 435.0 as you might have expected. Otherwise, the value of n would have been 435. So g must have had a different value. Give three possible of values for g that would have generated the observed output.

e. In the program from class and those in questions #4, #5 and #6, thinking of the int datatype as a perfect abstraction for an integer was responsible for the “surprising” results observed. This suggested that the int datatype is an imperfect abstraction. In this program, the variables f and g have the datatype double.

Explain in a few sentences how this example suggests that the double datatype is also an imperfect abstraction.

The point of these examples has in part been to try to motivate the importance for you, as a computer scientist, to learn about the lower levels of detail (i.e. the hidden stuff) that underlies the abstractions that we use.

🏆 8. Reflect on the above examples and write a few sentences about why it would be important for a Java programmer to understand some of the “irrelevant” details behind the abstractions that this language uses. Note: While we observed these behaviors using Java, they are not particular to Java. These specific examples can be observed most other programming languages, and all programming languages use abstractions that can lead to surprising behavior.

**Circuits and TINKERCAD**

In the first several labs in this course will explore the hardware abstraction hierarchy. We will see how electrical signals represent 0’s and 1’s, how transistors make logic gates, and how logic gates make circuits, and how circuits perform useful computation and make memory. We will be using a program named TINKERCAD to simulate the experience of building circuits from electronic components.

9. Use the link and the information in Moodle to Join our class on TINKERCAD.

There is no answer required here for this question. I will be able to see that you joined the class on TINKERCAD.

10. Watch the video *Introduction to Tinkercad - basic circuits and orientation* (Linked below). This video will introduce you to the basics of the TINKERCAD interface including how to create components, connect them together and simulate the circuit.

* <https://www.youtube.com/watch?v=lgWGUWvQ8Xc> (6:48)

a. Complete the following steps using TINKERCAD

i. Create a new Circuit in your TINKERCAD account.

ii. Change its name to Motor.

iii. Build the circuit that was shown the video.

iv. Simulate the circuit to ensure that it works.

Paste a *screen shot* of your circuit below as the answer to this question.

* + If you are not familiar with how to take screen shots the page *Capture the contents of a window or screen in Windows or macOS* from Indiana University gives a quick overview:
    - <https://kb.iu.edu/d/afws>
  + Notes:
    - Please capture only the relevant part of your screen to make your homework easier to read and grade.
    - **Use the “Paste and Match Formatting” option from the “Edit” menu to get the image to paste inside of the box.**

b. Reverse the connections between the battery and the motor so that the motor terminal that was connected to the positive battery terminal is now connected to the negative battery terminal and vice versa. What changes when you simulate the circuit?

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.