**CSCI 2302**

**Abstract Classes & Interfaces Chapter**

**Cloneable Lab**

Contents

[Intro 1](#_Toc85805850)

[Notes 1](#_Toc85805851)

[Shallow copy 3](#_Toc85805852)

[Deep copy 3](#_Toc85805853)

[Learning Goals 4](#_Toc85805854)

[Task 4](#_Toc85805855)

[Submit 5](#_Toc85805856)

Intro: Being able to make an exact copy of an object can be useful when you want to do something with an object but not lose the integrity of the data. This concept is how forensics use electronic devices; exact copies are made of the electronic device and then the investigation is performed on the clone.

Notes: The Cloneable interface specifies that an object can be cloned (an exact copy of the original). An object/class that implements the Cloneable interface has to implement/define the clone method.

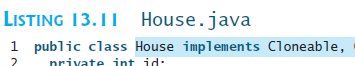


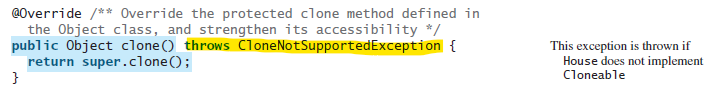
As you can see, the Cloneable interface is empty; this is referred to as a marker interface. A marker interface does not contain any states (constant variables/fields) nor abstract methods. It is simply a “marker “that denotes that a class possesses certain properties/behaviors.

In this case, implements Cloneable means that an object/a class can clone (or copy) its objects.

There is something we programmers ***HAVE*** to do in order to implement/define the clone method, we have to include throws CloneNotSupportedException in the clone method header *and* in the method that will have the invoking statement.

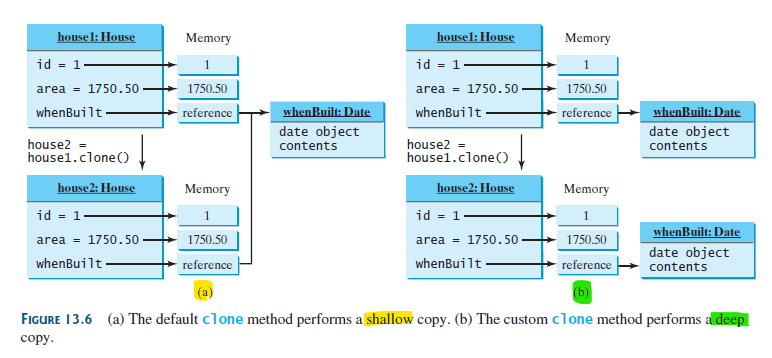
Example in use:





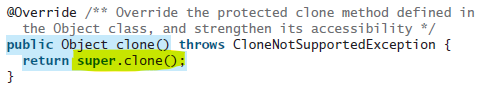
There is an import concept we have to understand when we implement/define the clone method. The difference between a shallow copy and a deep copy.

Let’s review the difference between a primitive data type and a reference data type. A primitive data type stores the value in the method’s memory. A reference data type stores the location of where its states and behaviors are stored elsewhere in memory. With that in mind:



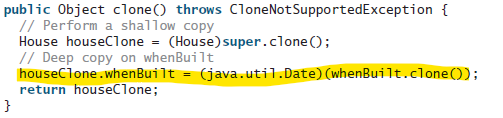
Let’s look at the code for the difference:

Shallow copy:



This works for primitive data types, but not for the reference data types; the reference data type just points to the original’s value.

Deep copy:



This is getting the reference data type value and assigning the value – not pointing to the original object.

How to use the clone method:



Note that you have to cast!

Learning Goals: To learn how to use the Cloneable interface and implement/define the clone method. To learn the difference between the shallow copy and the deep copy.

Task: Complete the steps outlined below:

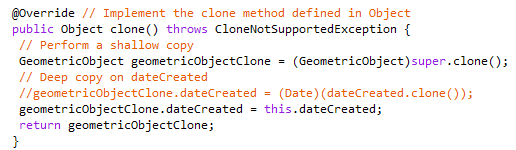
1. Download (please note: this version is not abstract – remember that you cannot instantiate an object from an abstract class) GeometricObject.java and implement the Cloneable interface in the GeometricObject class, then override the clone method; you have to perform a deep copy.

To do so, add to the class declaration:

public class ClassName implements Cloneable{

i.e. 

Then override the clone method:



1. Implement a mysfausername\_Cloneable\_Lab.java to complete the following (don’t forget about the CloneNotSupportedException in the main method header):
   1. Instantiate one GeometricObject object with a color and filled state and invoke the toString method.



* 1. Clone the object from a to another GeometricObject (don’t forget to cast!) and invoke the toString method.

It is always a good idea to test that an object is cloneable, prior to cloning it!



* 1. Change the states of filled and color of the cloned object.



* 1. Invoke the toString method of each object to see the deep copy.



Submit: Submit the mysfausername\_Cloneable\_Lab.java and GeometricObject.java in the Dropbox in Brightspace by D2L.