COSC 121: Review Exercises on Inheritance and Interfaces

1. Match each of the three relationships on the left to one terminology on the right:

$$\begin{tikzcd}[column sep=huge]
\text{HAS-A (e.g., a Library has a Book)} \arrow[r, dash] \arrow[dr, dash] & \text{Inheritance} \\
\text{IS-A (e.g., a Car is a Vehicle)} \arrow[ur, dash] & \text{Aggregation} \\
\text{USES-A (e.g., a Dog uses a Scanner)} \arrow[r, dash] & \text{Dependency}
\end{tikzcd}$$

2. What are some of the benefits of software reuse? List at least one that relates to the organization of your program, and one that relates to the maintenance of your program.

   - less code to write / cleaner

   - easier to debug / maintain

3. Why are the constructor methods in a parent class not inherited by a child class?

   constructors are for setting up an object

   doesn't make sense for child class to setup parent object

4. In the following example, list all three IS-A relationships defined:

```
public class Mammal { … }
public class Bird extends Mammal { … }
public class Penguin extends Bird { … }
```

   P is a M
   P is a B
   B is a M

5. What problem arises if multiple inheritance were allowed in Java? Give an example to illustrate this situation. collision

   if 2 "parents" have the same attribute / method who will the child inherit from?

6. What is something you can do by extending a class that you cannot do by importing it instead? modify it

   access protected info

7. What is the root class of every class hierarchy? Object class

8. Suppose you defined a `Dog` class and created two `Dog` objects. What criteria should be used to determine if the two `Dog` objects are equal or not? ex name, age, size, breed

   or just Id tag if available

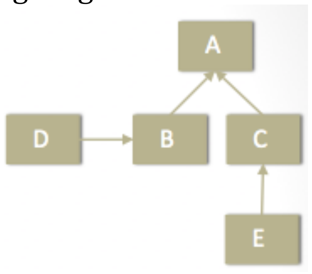
9. If you were to define the `equal()` method from Question 8, in which class would you need to define it? Dog

10. Define the `equal()` method you suggested in Question 8.

```
public boolean equal( Object other)
{
    Dog d2 = ( Dog) other;
    if ( idTag == d2 getIdtag () )
        return true;
    else
        return false.
}
```

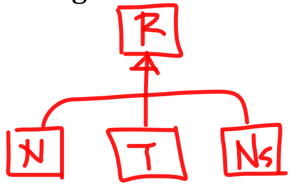
COSC 121: Review Exercises on Inheritance and Interfaces

11. Fill in the blanks using the following diagram:



a. Class A is a super/parent class of class C.
b. Class B is a sibling class of class C.
c. Class E is a grandchild class of class A.
d. Class D is a sub/child class of class B.
e. Class A is a grandparent/ ancestor class of class D.

12. Draw the inheritance diagram for the following classes: `Reading`, `Newspaper`, `Novel`, `Textbook`.



13. Based on your diagram from Question 12, which class(es) may make sense to be defined as an abstract class? Why? $\boxed{R}$ if don't need to create objects of it and want children to have default behaviour

14. When we define a method inside a class, how is an abstract method different from a non-abstract method? abstract methods 1 don't have body definitions
2 are declared abstract via reserved word abstract

15. List the differences between overriding and overloading.

- Overriding:
  in parent and children classes
  have same signature

- Overloading:
  In same class
  have different signatures

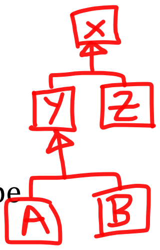
16. List the differences between inheritance and interfaces.

- Inheritance:
  defines when a class is a kind of another class
  a class can only inherit from one other class
  has attributes and methods

- Interfaces:
  defines when different/unrelated classes need to communicate in a standardized way
  a class can implement one or more interfaces
  has constants and abstract methods

17. List the differences between abstract classes and interfaces.
    - Abstract classes:
      can have attributes
      methods may be abstract or not
      methods can have different visibility
    - Interfaces:
      no attributes
      all methods are abstract
      all methods are public

18. True or false?
    T a. Implementing interfaces is good object-oriented programming practice.
    T b. Any class can extend from any other class.
    T c. A parent class is the same as a super class, and a child class is the same as a subclass.
    T d. A child class may define a method with the same name as a method in the parent class.
    F e. A child class can override the constructor of the parent class.
    T f. A child class cannot override a final method of the parent class.
    F g. It is considered poor design when a child class overrides a method from the parent class.
    T h. A child class may define an attribute with the same name as an attribute in the parent class. part of overriding
    T i. To reference a parent method, the child class can use `super()` if it is the constructor method it wants to call, or `super.methodName()` if it is any other method it wants to call.
    F j. To reference a parent attribute, the child class can use `super.attributeName` directly. attributes are private
    F k. Overloading is the same as overriding.
    T l. Classes Y and Z are children of X, and classes A and B are children of Y. If A, B, Y, and Z all have the same attribute var1, then var1 should be declared in X and inherited into Y, Z, A, and B.



    F m. A private method cannot be modified by a subclass. So if I want this method to be modifiable by subclasses, I must change its visibility to protected. ↳ or public
    F n. All methods in an abstract class must be abstract.
    F o. An abstract method can be defined as final.
    T p. An abstract method cannot be defined as static.
    F q. An abstract method must be have public visibility. protected is okay
    F r. Abstract methods are used to define classes that have no constructors.
    F s. A class that inherits from an abstract class must define all the inherited abstract methods. it can opt to declare those as abstract instead
    F t. An interface is a special type of class. an interface is a set of abstract methods
    T u. A class can implement as many interfaces as it wants.

T v. All methods in an interface must be abstract.
T w. An interface does not have any attributes.
T x. Just like an abstract class, an interface cannot be instantiated.
T y. A class that implements an interface must define all the implemented abstract methods.
T z. All the methods in an interface must be public.
F aa. Interfaces cannot be extended.
F bb. The method `compareTo()` is an inherited method available from the `Comparable` class. ↳ implemented '
T cc. Any class can define its own `compareTo()` method by implementing the `Comparable` interface. In that case, the definition must follow the same signature as dictated by the `Comparable` interface.
T dd. Using an interface is a better choice than using an abstract class when you want other classes to conform to a standard set of method protocols.
T ee. Using an abstract class is a better choice when you want other classes to have some default behavior.

19. When we first introduce shapes to children, there are typically three shapes we tell them about: square, circle, and triangle. From these basic shapes, other shapes are derived. Now, consider the classes `Circle` and `Oval` that share common attributes and methods. Would you relate these two via an inheritance relationship or an interface relationship? Why? inheritance

an Oval is a kind of Circle

20. Imagine a game in which players can attack other players' game elements, such as boxes, mirrors, balloons, etc. Each type of game element belongs to a different class. For example, there is a `Box` class, where a series of `Box` objects can be created; there is a `Mirror` class, etc., there is a `Balloon` class, etc.

What these game elements have in common is that they all have a `break()` method that explains what a player needs to do in order for the game element to break. They also all have a `isBroken()` method that returns a Boolean depending on whether the element is intact or broken at the time.

Should the game element classes, `Box`, `Mirror`, `Balloon`, be related via inheritance or interface? Why? interf ce

a Box/Mirror/Balloon is not a kind of a Box/Mirror/Balloon

21. Continuing with Question 20. Suppose `break()` and `isBroken()` are abstract methods initially defined in `Breakable`. Assuming it only has these two methods, define `Breakable` fully.

```
public interface Breakable
{
    public void break(),
    public boolean isBroken(),

}
```

22. Consider the superclass `Square` and the subclass `Rectangle`. The `Square` class has attributes `numSides` and `length`, as well as the following methods:
    - `calcPerimeter()`, which returns a double based on the standard definition of the perimeter of a square
    - `calcArea()`, which returns a double based on the standard definition of the area of a square
    - `isBiggerThan()`, which returns a Boolean that indicates if the object itself is bigger in area than an input Square `object`

    If you had to define the `Rectangle` class:

    a. What attributes will it have? length, width

    b. What methods will it have? same as Square

    c. What will be the visibility for each? attributes all private
    methods all public because I want unrelated classes to be able to call them

    d. Which inherited method will you override?
    calcPerimeter() and calcArea()

    Practice by defining both classes fully.