COMI 2510 Advanced Programming and Design
Lesson 3: Inheritance and Polymorphism
Lab

1. Complete this on your own, then review the videos and code in the class website: Ship, CruiseShip and CargoShip classes, programming challenge #10 from Chapter 10.

Design a `Ship` class that has the following members:

- A field for the name of the ship (a `String`)
- A field for the year that the ship was built (a `String`)
- A constructor and appropriate accessors and mutators
- A `toString` method that displays the ship's name and the year it was built.

Design a `CruiseShip` class that extends the `Ship` class. The `CruiseShip` class should have the following members:

- A field for the maximum number of passengers (an `int`)
- A constructor and appropriate accessors and mutators
- A `toString` method that overrides the `toString` method in the base class. The `CruiseShip` class's `toString` method should display only the ship's name and the maximum number of passengers.

Design a `CargoShip` class that extends the `Ship` class. The `CargoShip` class should have the following members:

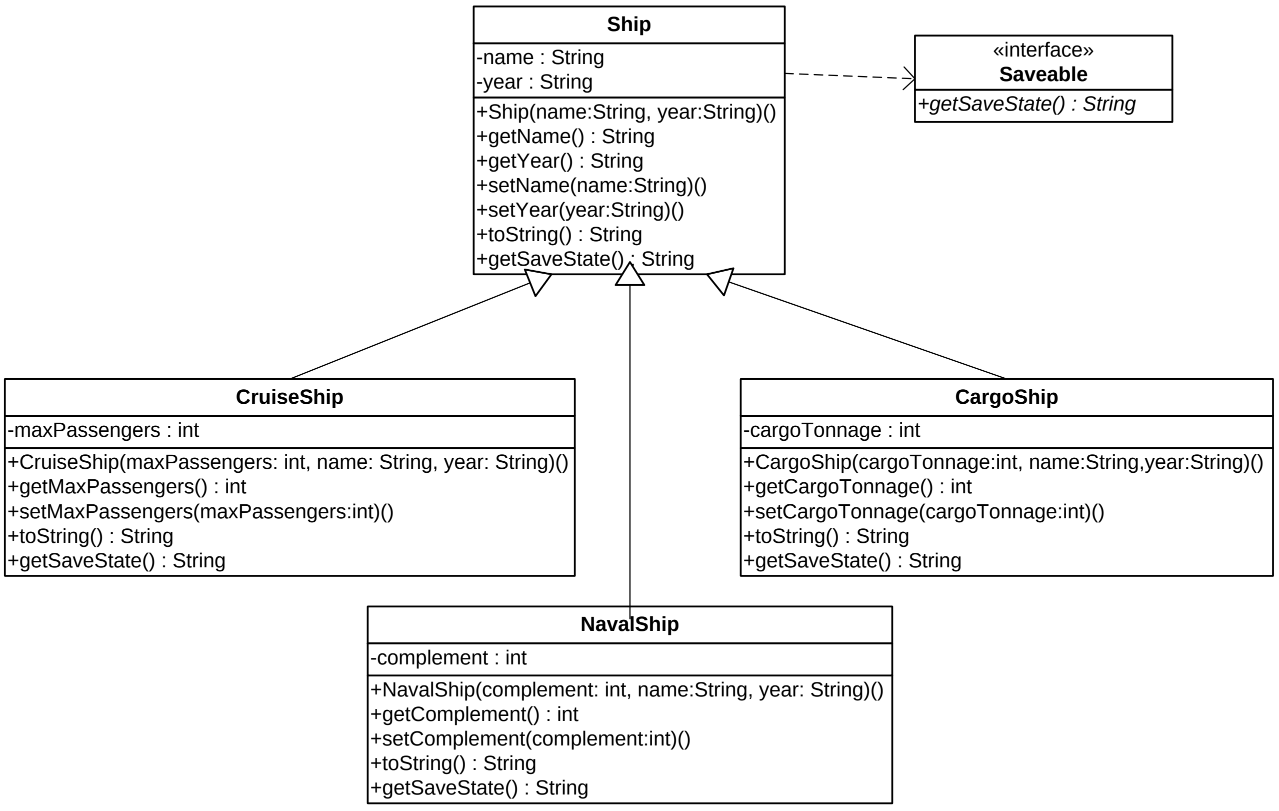
- A field for the cargo capacity in tonnage (an `int`)
- A constructor and appropriate accessors and mutators
- A `toString` method that overrides the `toString` method in the base class. The `CargoShip` class's `toString` method should display only the ship's name and the ship's cargo capacity.

Draw class diagrams for the classes and then code them.

Demonstrate the classes in a program that has a `Ship` array. Assign various `Ship`, `CruiseShip`, and `CargoShip` objects to the array elements. The program should then step through the array, calling each object's `toString` method.

2. On your own, enhance the program as outlined in this class diagram.

- Add another subclass, `NavalShip`, which has a field, accessor, and mutator for the ship's complement. Override `toString` to display the ship's name and complement.
- Implement the `Saveable` interface. This is an interface that any object that can be saved to a file should implement (in our imaginary software).
- Write the `getSaveState` method (from the interface `Saveable`) for each class. For each class, the `getSaveState` method should return a `String` of state information about an object. This should consist of the class name and all fields, separated by a #. Note that for subclasses, the superclass information should also be included.
  Here is an example return value from `getSaveState` for an object of class `Ship`:
  Ship#Enterprise#2245
  Here is an example return value from `getSaveState` for an object of class `CargoShip`:
  CargoShip#Sulaco#1979#20000
- Rewrite the main method to include ships of the new type and to invoke not only `toString` for each instance but also `getSaveState`.


Ship
-name : String
-year : String
+Ship(name:String, year:String)()
+getName() : String
+getYear() : String
+setName(name:String)()
+setYear(year:String)()
+toString() : String
+getSaveState() : String
«interface»
Saveable
+getSaveState() : String
CruiseShip
-maxPassengers : int
+CruiseShip(maxPassengers: int, name: String, year: String)()
+getMaxPassengers() : int
+setMaxPassengers(maxPassengers:int)()
+toString() : String
+getSaveState() : String
CargoShip
-cargoTonnage : int
+CargoShip(cargoTonnage:int, name:String,year:String)()
+getCargoTonnage() : int
+setCargoTonnage(cargoTonnage:int)()
+toString() : String
+getSaveState() : String
NavalShip
-complement : int
+NavalShip(complement: int, name:String, year: String)()
+getComplement() : int
+setComplement(complement:int)()
+toString() : String
+getSaveState() : String