Unit 6: OOP & TypeScript: Exercise 2 – Shipping Containers (Classes, Interfaces)

Assigned: N/A

Due: N/A

Completed: TBD

**LMS:** <https://lms.grandcircus.co/course/view.php?id=231&section=4>

**Google Doc:** <https://docs.google.com/document/d/1vkorSTXMLZIG9DcycI0mj8dJtYIVkCbg1ZWisFqIcyA/preview>

**GitHub:** N/A

**Overview:** Using TypeScript, create and test interfaces and classes that can be used to manage loading shipping containers onto trucks and ships. This lab does not represent a complete program; these pieces could be used as part of a larger TypeScript program.

**Build Specifications**

* You will create a ShippingContainer interface and two classes that implement that interface: LightContainer and HeavyContainer.
* You will create a Transporter interface and two classes that implement that interface: Truck and Ship.
* Each class has required Jest test cases that you must write.
* You will also create three functions that work with these interfaces and classes.
* All three functions are written in one file and also have required Jest test cases.

**ShippingContainer Interface**

In its own file, create an interface named ShippingContainer and export it.

* Properties:
  + destination (a string)
  + cargoWeight (a number)
* Methods:
  + getGrossWeight: Has no parameters. It returns a number.
* Jest Tests: N/A - We won't be writing tests directly for our interfaces.

**LightContainer Class**

In its own file, create a class named LightContainer and export it.

* Implements the ShippingContainer interface.
* Properties:
  + destination (a string)
  + cargoWeight (a number) - the weight of the cargo inside the container
* Constructor Parameters:
  + destination (a string) sets the destination property
  + cargoWeight (a number) sets the cargoWeight property. This parameter is optional and has a default value of **0**.
* Methods:
  + getGrossWeight: Has no parameters. Simply returns the cargoWeight.
* Jest Tests:
  + The destination and cargoWeight properties are set from the constructor parameters.
  + cargoWeight defaults to **0**, when the second constructor parameter is omitted.
  + getGrossWeight returns the cargoWeight (write 2 test cases with different cargoWeights)

**HeavyContainer Class**

In its own file, create a class named HeavyContainer and export it.

* Implements the ShippingContainer interface.
* Properties:
  + tareWeight (a number) - this is the weight of the container when empty
  + destination (a string)
  + cargoWeight (a number) - the weight of the cargo inside the container
* Constructor Parameters:
  + tareWeight (a number) sets the tareWeight property
  + destination (a string) sets the destination property
  + cargoWeight (a number) sets the cargoWeight property. This parameter is optional and has a default value of **0**.
* Methods:
  + getGrossWeight: No parameters. Returns the tareWeight plus cargoWeight.
* Jest Tests:
  + The tareWeight, destination, and cargoWeight properties are set from the constructor parameters.
  + cargoWeight defaults to **0**, when the third constructor parameter is omitted.
  + getGrossWeight returns the tareWeight plus the cargoWeight (write 2 test cases with different tareWeights and cargoWeights)

**Transporter Interface**

In its own file, create an interface named Transporter and export it.

* Properties:
  + maxWeight (a number)
* Methods:
  + addContainer: Takes one parameter: container (a ShippingContainer). It returns nothing.
  + getTotalWeight: Has no parameters. It returns a number.
  + isOverweight: Has no parameters. It returns a Boolean.
* Jest Tests: N/A - We won't be writing tests directly for our interfaces.

**Truck Class**

In its own file, create a class named Truck and export it.

* Implements the Transporter interface.
* Properties:
  + maxWeight (a number) - the maximum cargo capacity
  + container (a ShippingContainer or null) - ALWAYS STARTS as null - the container currently loaded on the truck
* Constructor Parameters:
  + maxWeight (a number) sets the maxWeight property
* Methods:
  + addContainer: Takes one parameter: container (a ShippingContainer). Sets the container property. It returns nothing.
  + getTotalWeight: Has no parameters. Returns the gross weight of the container property using getGrossWeight. Or if container is null, returns **0**.
  + isOverweight: Has no parameters. Calls getTotalWeight and returns **true** if the result is greater than maxWeight, **false** otherwise.
* Jest Tests:
  + The maxWeight property is set from the constructor parameter.
  + The container property is set to **null** in a new Truck instance.
  + Calling addContainer sets the container property.
  + getTotalWeight returns the gross weight of the container when a container is added.
    - Try it several times with different LightContainers and HeavyContainers.
  + getTotalWeight returns **0** when container is **null**.
  + isOverweight returns **true** when the total weight is greater than maxWeight.
  + isOverweight returns **false** when the total weight is less than maxWeight.
  + isOverweight returns **false** when the total weight is equal to maxWeight.

**Ship Class**

In its own file, create a class named Ship and export it.

* Implements the Transporter interface.
* Properties:
  + maxWeight (a number) - the maximum cargo capacity
  + containers (an array of ShippingContainer) - ALWAYS STARTS as an empty array - the containers currently loaded on the ship
* Constructor Parameters:
  + maxWeight (a number) sets the maxWeight property
* Methods:
  + addContainer: Takes one parameter: container (a ShippingContainer). Adds the container to the containers array property. It returns nothing.
  + getTotalWeight: Has no parameters. Returns the sum of the gross weights of all the containers in the containers array property by calling getGrossWeight for each one. Or if containers is empty, returns **0**.
  + isOverweight: Has no parameters. Calls getTotalWeight and returns **true** if the result is greater than maxWeight, **false** otherwise.
* Jest Tests:
  + The maxWeight property is set from the constructor parameter.
  + The containers property is set to an empty array in a new Ship instance.
  + Calling addContainer adds to the containers array property.
  + Calling addContainer twice adds both containers to the containers array property.
  + getTotalWeight returns the combined gross weight of the containers in the array.
    - Try it several times with a mix of LightContainers and HeavyContainers added.
  + getTotalWeight returns **0** when containers is empty.
  + isOverweight returns **true** when the total weight is greater than maxWeight.
  + isOverweight returns **false** when the total weight is less than maxWeight.
  + isOverweight returns **false** when the total weight is equal to maxWeight.

**findContainersByDestination Function**

In the functions.ts file, create a function named findContainersByDestination and export it.

* Parameters:
  + containers (an array of ShippingContainer)
  + destination (a string)
* Returns: an array of ShippingContainer
* Functionality: Filter the containers array to find only the containers that have a matching destination. Return a new array of those matching containers.
* Jest Tests: For each test case, create an array of ShippingContainers. Call findContainersByDestination with this array and a destination and confirm the correct result.
  + Do a test case with an array of LightContainer.
  + Do a test case with an array that has a mix of LightContainer and HeavyContainer.
  + Do a test case where none of the containers match the destination. (Expect an empty array as the result.)
  + Do a test case with an empty array. (Expect an empty array as the result.)

**findOverweightTransporters Function**

In the functions.ts file, create a function named findOverweightTransporters and export it.

* Parameters:
  + transporters (an array of Transporter)
* Returns: an array of Transporter
* Functionality: Filter the transporters array to find only the transporters that return true for isOverweight. Return a new array of those matching transporters.
* Jest Tests: For each test case, create an array of Transporters. Call findOverweightTransporters with this array and a destination and confirm the correct result.
  + Do a test case with an array of Trucks, some overweight, some not.
  + Do a test case with an array that has a mix of Truck and Ship, some overweight, some not.
  + Do a test case with an array of Transporters where none are overweight. (Expect an empty array as the result.)
  + Do a test case with an empty array. (Expect an empty array as the result.)

**isSafeToAddContainer Function**

In the functions.ts file, create a function named isSafeToAddContainer and export it.

* Parameters:
  + ship (a Ship)
  + container (a ShippingContainer)
* Returns: a Boolean
* Functionality: Determine if the ship would be overweight if the container were added. (Is ship total weight + container gross weight less than or equal to the ship max weight?) If so, return **true** (it is safe). Otherwise, return **false** (it is not safe).
* Jest Tests:
  + isSafeToAddContainer returns **true** for an empty ship and empty LightContainer when transporter maxWeight is 5000.
  + isSafeToAddContainer returns **true** for an empty ship and a LightContainer with some cargo, but less than maxWeight.
  + isSafeToAddContainer returns **true** for an empty ship and a HeavyContainer with some cargo, but less than maxWeight.
  + isSafeToAddContainer returns **false** for an empty ship and a LightContainer with some cargo, more than maxWeight.
  + isSafeToAddContainer returns **false** for an empty ship and a HeavyContainer with some cargo, more than maxWeight.
  + isSafeToAddContainer returns **true** for an empty ship and a container with the same gross weight as the maxWeight.
  + Create a ship with one or more containers already added. isSafeToAddContainer returns **true** for a container that is light enough to be added to this ship.
  + Create a ship with one or more containers already added. isSafeToAddContainer returns **false** for a container that is too heavy to be added to this ship.