**Exercise Set 1: Working with Interfaces**

**Exercise 1.1: Basic Interface**

* **Objective**: Create a simple interface and implement it.
* **Task**: Define an Animal interface with the following properties and methods:
  + species: a string representing the species of the animal.
  + age: a number representing the age of the animal.
  + makeSound(): a method that returns a string indicating the sound the animal makes.
* **Challenge**: Create an object of type Animal for a Dog with appropriate values.

**Exercise 1.2: Extending Interfaces**

* **Objective**: Practice interface extension.
* **Task**: Extend the Animal interface to create a Pet interface with additional properties:
  + name: a string representing the name of the pet.
  + ownerName: a string representing the pet's owner's name.
* **Challenge**: Create an object of type Pet for a Cat with all properties filled out.

**Exercise 1.3: Implementing Interfaces with Functions**

* **Objective**: Use interfaces to type-check functions.
* **Task**: Create a Book interface with properties title (string), author (string), and price (number).
* **Additional Task**: Write a function logBookInfo that accepts an argument of type Book and logs the book's details.

**Exercise Set 2: Creating Classes**

**Exercise 2.1: Basic Class with Constructor**

* **Objective**: Create a simple class with properties and methods.
* **Task**: Define a Car class with the following properties and methods:
  + make: string (brand of the car)
  + model: string (model of the car)
  + year: number (year of manufacture)
  + start(): method that logs Car started.
* **Challenge**: Create an instance of Car for a Toyota Corolla manufactured in 2020 and call the start() method.

**Exercise 2.2: Access Modifiers**

* **Objective**: Work with public and private access modifiers.
* **Task**: Update the Car class:
  + Make the make and model properties public.
  + Make the year property private.
  + Add a method getCarAge() that calculates the age of the car based on the current year.
* **Challenge**: Try to access the year property directly from an instance. Observe what happens.

**Exercise 2.3: Class Inheritance**

* **Objective**: Practice creating a subclass that extends a superclass.
* **Task**: Define a Vehicle class with a startEngine() method that logs "Engine started".
  + Create a subclass Motorcycle that extends Vehicle and adds a method revEngine() that logs "Revving engine".
* **Challenge**: Create an instance of Motorcycle and call both startEngine() and revEngine().

**Exercise Set 3: Combining Interfaces and Classes**

**Exercise 3.1: Implementing an Interface in a Class**

* **Objective**: Implement an interface in a class.
* **Task**: Define an Employee interface with the following properties:
  + id: number
  + name: string
  + getSalary(): method that returns a number
* **Next Task**: Create an Engineer class that implements Employee with an additional property:
  + specialization: string (e.g., "Software", "Electrical")
  + Implement the getSalary() method to return a base salary.
* **Challenge**: Create an instance of Engineer and call the getSalary() method.

**Exercise 3.2: Using Abstract Classes with Interfaces**

* **Objective**: Combine an abstract class with an interface.
* **Task**: Define an interface Appliance with a turnOn() method that logs "Appliance is now on".
  + Create an abstract class Device that implements Appliance.
  + Add a brand property and a method displayBrand() in Device.
  + Create a concrete class WashingMachine that extends Device and calls both turnOn() and displayBrand() in its own startWashing() method.
* **Challenge**: Instantiate WashingMachine, call startWashing(), and observe the output.

**Exercise 3.3: Multiple Interfaces**

* **Objective**: Practice implementing multiple interfaces in one class.
* **Task**: Define two interfaces:
  + Flyable with a fly() method.
  + Swimmable with a swim() method.
  + Create a class Duck that implements both interfaces and defines both methods.
* **Challenge**: Instantiate Duck and call both fly() and swim() methods.

**Exercise Set 4: Advanced Challenges**

**Exercise 4.1: Interface with Optional Properties**

* **Objective**: Work with optional properties in interfaces.
* **Task**: Create a Product interface with name (string), price (number), and an optional discount (number).
* **Challenge**: Write a function calculateFinalPrice that takes a Product and returns the price after applying the discount (if available).

**Exercise 4.2: Interface for Function Types**

* **Objective**: Use interfaces to define function types.
* **Task**: Create an interface Operation that defines a function type that accepts two numbers and returns a number.
* **Next Task**: Define two functions add and multiply that conform to the Operation interface.
* **Challenge**: Write a calculate function that takes two numbers and an Operation (like add or multiply) and returns the result.

**Exercise 4.3: Abstract Class and Interface Combined Challenge**

* **Objective**: Practice advanced TypeScript concepts together.
* **Task**: Define an interface Trainable with a train() method.
  + Create an abstract class Athlete with properties name and sport.
  + Create a Swimmer class that extends Athlete and implements Trainable.
  + In Swimmer, implement train() to log "Swimming training started for [name]".
* **Challenge**: Instantiate Swimmer and call train() to observe the functionality.