Advanced Travel Planner System using ES6 Classes & OOP Principles

Objective: Develop an in-depth travel planner system that uses the principles of OOP. This system will empower users to meticulously plan trips across multiple destinations with diverse transportation modes, lodging choices, and activities.

---

1. Abstraction:

- Begin with the foundation class `Destination`:
  - **Properties**:
    - `name`: Name of the destination.
    - `description`: A brief description.
    - `highlights`: An array of places or points of interest.
    - `#weather`: A private property for storing weather information.
  - **Methods**:
    - `addHighlight(place)`: Adds a new highlight, but only if it hasn't been added before.
    - `displayInfo()`: Prints out the details of the destination.
    - `destinationType()`: A basic implementation to return "This is a basic destination."

---

2. Encapsulation:

- Protect the integrity of your data:
  - The `#weather` attribute in `Destination` is private. Ensure it can't be changed directly.
- Create a getter and sette method for the `#weather` attribute:
  - `get wather()`: A getter that returns the current weather.
  - `set weather(weather)`: A setter for the `#weather` property that only accepts the following values: "Sunny", "Rainy", "Cloudy", "Windy", "Stormy". If an invalid weather type is provided, log an error message to the console.

---

3. Inheritance:

- Create specialized subclasses that inherit from `Destination`:
    - **City**:
        - **Properties**:
            - `population`: Number of residents.
            - `publicTransitOptions`: Array of available public transit methods.
        - **Methods**:
            - `cityFacts()`: Prints out the city-specific information.
        - Override `displayInfo()` to first show the basic destination details and then the city-specific details.
        - Override `destinationType()` to return "This is a vibrant city."
    - **NaturalReserve**:
        - **Properties**:
            - `flora`: Array of plant species found.
            - `fauna`: Array of animal species observed.
        - **Methods**:
            - `wildlifeSpotting()`: Prints out information on flora and fauna.
        - Update `displayInfo()` to show both general and nature-specific details.
        - Override `destinationType()` to say "This is a serene natural reserve."
    - **Beach**:
        - **Properties**:
            - `sandType`: Description of the sand (e.g., "fine white").
            - `waterClarity`: Clarity of the water (e.g., "crystal clear").
        - **Methods**:
            - `beachActivities()`: Presents attributes specific to the beach.
        - Customize `displayInfo()` to encapsulate both general details and beach-specific attributes.
        - Modify `destinationType()` to return "This is a beautiful beach."

---

4. Polymorphism:

- Ensure that each destination subclass has its unique `displayInfo()` method that shows specific information suited to its type.
- Similarly, each subclass should provide its interpretation of `destinationType()`.

---

5. Associations & Composition:

- Develop a class called `Trip`:
  - **Properties**:
    - `name`: Title of the trip.
    - `duration`: Duration in days.
    - `destinations`: An array of `Destination` objects.
    - `transportation`: An array for travel modes.
    - `lodging`: An array of places to stay.
  - **Methods**:
    - `addDestination(destination)`: Inserts a destination to the trip if it's an instance of `Destination`.
    - `setTransportation(transport)`: Sets the mode of transport.
    - `setLodging(lodge)`: Specifies the lodging option.

---

6. Advanced Polymorphism & Dynamic Behavior:

- Implement `Transport` and `Lodging` classes:
  - **Transport**:
    - **Properties**:
      - `type`: Kind of transport (e.g., `Train`, `Flight`).
      - `details`: Specific details about the transport (e.g., make and model of the car or airplane).
  - **Lodging**:
    - **Properties**:
      - `type`: Type of accommodation (e.g., `Hotel`, `Hostel`).
      - `amenities`: An array of available amenities (e.g., ["Pool", "WiFi"]).

Tasks:

1. Implement the `Destination` class and its methods, ensuring they perform as described.
2. Develop the subclasses, ensuring each adds its unique properties and methods.
3. Demonstrate encapsulation: ensure the weather information is only accessible through the `get weather()` getter method and can be safely updated via `set weather(weather)` setter method.
4. Exhibit polymorphism by designing various destinations, updating their highlights, and displaying relevant information.
5. **Tests**: Run the tests in the tests directory to verify your code runs correctly.