Exercise 1: Product and Inventory Management

Problem Description:

You need to create two classes: `Product` to represent items with a name, price, and quantity, and `Inventory` to manage a collection of these products. The `Inventory` class should be able to calculate the total value of all products and identify products with low stock.

Requirements:

1. Create a class `Product` with a constructor that takes `name` (string), `price` (number), and `quantity` (number).
2. Add a method `getTotalPrice()` to the `Product` class that returns the product of its price and quantity.
3. Create a class `Inventory` with a constructor that initializes an empty array to hold `Product` objects.
4. Add a method `addProduct(product)` to `Inventory` that adds a `Product` object to its internal array.
5. Add a method `getTotalInventoryValue()` to `Inventory` that loops through all products in the inventory and returns the sum of their total prices (using the `getTotalPrice` method of each product).
6. Add a method `listLowStockProducts(threshold)` to `Inventory` that takes a number `threshold`. It should iterate through the products and print the `name` of any product whose `quantity` is less than or equal to the `threshold`.

Input:

```
// Assuming Product and Inventory classes are defined as per requirements

const inventory = new Inventory();
const product1 = new Product("Laptop", 1200, 5);
const product2 = new Product("Mouse", 25, 50);
const product3 = new Product("Keyboard", 75, 10);
const product4 = new Product("Monitor", 300, 3);

inventory.addProduct(product1);
inventory.addProduct(product2);
inventory.addProduct(product3);
inventory.addProduct(product4);

console.log("Total Inventory Value:", inventory.getTotalInventoryValue());
inventory.listLowStockProducts(5);

// Expected Output (logs):
// Total Inventory Value: 16900
// Products with quantity at or below 5: Laptop Monitor
```

Exercise 2: Student and Grade Calculation

Description:

Create a class `Student` to store a student's name and a list of their scores. The class should be able to calculate the student's average grade and determine if they are passing or failing based on a threshold.

Requirements:

1. Create a class `Student` with a constructor that takes `name` (string) and an array of numbers `scores`.
2. Add a method `getAverageGrade()` to the `Student` classs. This method should calculate the average of the numbers in the `scores` array. Handle the case where the `scores` array is empty by returning 0.
3. Add a method `getGradeStatus()` that returns "Passing" if the average grade (calculated by `getAverageGrade()`) is 60 or greater, and "Failing" otherwise
4. Create an array of `Student` objects
5. Loop through the array of students and print the name, average grade (formatted to two decimal places), and grade status for each student

Input:

```
// Assuming Student class is defined as per requirements

const student1 = new Student("Alice", [85, 90, 78, 92]);
const student2 = new Student("Bob", [55, 60, 45, 50]);
const student3 = new Student("Charlie", []);

const students = [student1, student2, student3];

console.log("Student Grade Report:");
// Loop through students and print their details - see the expected output
below
```

```
# Expected Output (logs):
Student Grade Report:
Alice: Average Grade - 86.25, Status - Passing
Bob: Average Grade - 52.50, Status - Failing
Charlie: Average Grade - 0.00, Status - Failing
```

Exercise 3: Bank Account with Transactions

Problem Description:

Design a `BankAccount` class that supports depositing, withdrawing, and keeping track of transactions.

Note: You might need to watch this video to better understand how to use Dates in JavaScript: Video Link

Requirements:

1. Create a class `BankAccount` with a constructor that takes an `initialBalance` and initializes an empty array `transactions` to store transaction details.
2. Add a method `deposit(amount)`. If `amount` is positive, add it to the `balance` and add an object `{ type: 'deposit', amount: amount, date: new Date() }` to the `transactions` array. Otherwise, print an error.
3. Add a method `withdraw(amount)`. If `amount` is positive and less than or equal to the `balance`, subtract it from the `balance` and add an object `{ type: 'withdrawal', amount: amount, date: new Date() }` to the `transactions` array. Otherwise, print "Insufficient funds" or "Invalid withdrawal amount".
4. Add a method `getBalance()` that returns the current `balance`.
5. Add a method `getTransactions()` that loops through the `transactions` array and prints the details of each transaction (type, amount, date).
6. Create a `BankAccount` object and perform a sequence of deposit and withdrawal operations.
7. Call `getBalance()` and `getTransactions()` to display the results.

Input:

```
// Assuming BankAccount class is defined as per requirements

const myAccount = new BankAccount(1000);

myAccount.deposit(500);
myAccount.withdraw(200);
myAccount.deposit(150);
myAccount.withdraw(1500); // Insufficient funds
myAccount.withdraw(0); // Invalid amount

console.log("\nCurrent Balance:", `$${myAccount.getBalance().toFixed(2)}`);
myAccount.getTransactions();
```

```
Deposited: $500.00. New balance: $1500.00
Withdrew: $200.00. New balance: $1300.00
Deposited: $150.00. New balance: $1450.00
Insufficient funds.
Invalid withdrawal amount. Amount must be positive.

Current Balance: $1450.00

Transaction History:
// Date and Time will vary
[Date] [Time] - DEPOSIT: $500.00
[Date] [Time] - WITHDRAWAL: $200.00
[Date] [Time] - DEPOSIT: $150.00```
```