Program 1:

import \* as readline from 'readline';

// Base class

class AccountHolder {

  protected name: string;

  protected accountNumber: string;

  constructor(name: string, accountNumber: string) {

    this.name = name;

    this.accountNumber = accountNumber;

  }

  displayDetails(): void {

    console.log(`Account Holder: ${this.name}`);

    console.log(`Account Number: ${this.accountNumber}`);

  }

// Derived class

class BankAccount extends AccountHolder {

  private balance: number;

  constructor(name: string, accountNumber: string, balance: number = 0) {

    super(name, accountNumber);

    this.balance = balance;

  }

  credit(amount: number): void {

    this.balance += amount;

    console.log(`Credited ₹${amount}. New Balance: ₹${this.balance}`);

  }

  withdraw(amount: number): void {

    if (amount > this.balance) {

      console.log("Insufficient balance.");

    } else {

      this.balance -= amount;

      console.log(`Withdrawn ₹${amount}. Remaining Balance: ₹${this.balance}`);

    }

  }

  checkBalance(): void {

    console.log(`Current Balance: ₹${this.balance}`);

  }

}

// CLI

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout,

});

const user = new BankAccount("Dhiran", "1234567890", 1000);

function showMenu(): void {

  console.log("\n--- Bank Menu ---");

  console.log("1. View Account Details");

  console.log("2. Credit");

  console.log("3. Withdraw");

  console.log("4. Check Balance");

  console.log("5. Exit");

  rl.question("Choose an option: ", handleMenu);

}

function handleMenu(option: string): void {

  switch (option) {

    case '1':

      user.displayDetails();

      showMenu();

      break;

    case '2':

      rl.question("Enter amount to credit: ", (amount) => {

        user.credit(parseFloat(amount));

        showMenu();

      });

      break;

    case '3':

      rl.question("Enter amount to withdraw: ", (amount) => {

        user.withdraw(parseFloat(amount));

        showMenu();

      });

      break;

    case '4':

      user.checkBalance();

      showMenu();

      break;

    case '5':

      console.log("Thank you for banking with us!");

      rl.close();

      break;

    default:

      console.log("Invalid option. Try again.");

      showMenu();

  }

}

showMenu();

op:

--- Bank Menu ---

1. View Account Details

2. Credit

3. Withdraw

4. Check Balance

5. Exit

Choose an option: 1

Account Holder: Dhiran

Account Number: 1234567890

--- Bank Menu ---

1. View Account Details

2. Credit

3. Withdraw

4. Check Balance

5. Exit

Choose an option: 5

Thank you for banking with us!

Program 2:

import \* as readline from 'readline';

interface Product {

  id: number;

  name: string;

  price: number;

}

const products: Product[] = [

  { id: 1, name: 'Laptop', price: 999 },

  { id: 2, name: 'Phone', price: 499 },

  { id: 3, name: 'Tablet', price: 299 },

];

const getProductById = (id: number): Product | undefined => {

  return products.find((product) => product.id === id);

};

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout,

});

rl.question('Enter the product ID to search: ', (input) => {

  const id = parseInt(input);

  const product = getProductById(id);

  if (product) {

    console.log(`Product found: ${product.name} - $${product.price}`);

  } else {

    console.log('Product not found.');

  }

  rl.close();

});

Op:

Enter the product ID to search: 1

Product found: Laptop - $999

Program 3:

class Queue {

    private items: any[] = [];

    // Enqueue: Add a value to the rear of the queue

    enqueue(value: any): void {

        this.items.push(value);

        console.log(`${value} is enqueued`);

        this.printQueue();

    }

    // Dequeue: Remove and return the value from the front of the queue

    dequeue(): any {

        if (this.isEmpty()) {

            console.log("Queue is empty, nothing to dequeue");

            this.printQueue();

            return undefined;

        }

        const value = this.items.shift();

        console.log(`${value} is dequeued`);

        this.printQueue();

        return value;

    }

    // isEmpty: Check if the queue is empty

    isEmpty(): boolean {

        return this.items.length === 0;

    }

    // Helper method to print the current queue

    private printQueue(): void {

        console.log(`Current queue: [${this.items.join(", ")}]`);

    }

}

// Example usage:

console.log("For Numbers!");

const queue\_numbers = new Queue();

queue\_numbers.enqueue(1);

queue\_numbers.enqueue(2);

console.log(queue\_numbers.dequeue());

console.log(queue\_numbers.isEmpty());

console.log(queue\_numbers.dequeue());

console.log(queue\_numbers.isEmpty());

console.log(queue\_numbers.dequeue());

console.log("\nFor Strings!");

const queue\_string = new Queue();

queue\_string.enqueue("bat");

queue\_string.enqueue("ball");

console.log(queue\_string.dequeue());

op:

For Numbers!

1 is enqueued

Current queue: [1]

2 is enqueued

Current queue: [1, 2]

1 is dequeued

Current queue: [2]

1

false

2 is dequeued

Current queue: []

2

true

Queue is empty, nothing to dequeue

Current queue: []

Undefined

For Strings!

bat is enqueued

Current queue: [bat]

ball is enqueued

Current queue: [bat, ball]

bat is dequeued

Current queue: [ball]

Bat