**Java Question Bank**

**Question 1.)**

You are developing a Java program for a simple library management system using an ArrayList to manage books in the library. Each book should have attributes including the title, author, and ISBN number. Implement a Book class with methods to perform the following actions:

1. Allow users to add a new book to the library.

2. Allow users to remove a book from the library by providing its ISBN number.

3. Enable users to search for a book by title and display its details.

4. Display the list of all books currently in the library.

**import java.util.ArrayList;**

**class Book**

**{**

**private String title;**

**private String author;**

**private int ISBN;**

**public Book(String title,String author,int ISBN) {**

**this.title=title;**

**this.author = author;**

**this.ISBN = ISBN;**

**}**

**public String getTitle(){**

**return title;**

**}**

**public String getAuthor() {**

**return author;**

**}**

**public int getIsbn() {**

**return ISBN;**

**}**

**public void display() {**

**System.*out*.println("title "+title +" Author "+author+" ISBN number: "+ISBN);**

**}**

**}**

**class Library {**

**private ArrayList<Book> books;**

**public Library() {**

**books = new ArrayList<>();**

**}**

**public void addBook(Book book) {**

**books.add(book);**

**System.*out*.println("Book added");**

**}**

**public boolean removeBook(int isbn) {**

**for(Book book : books) {**

**if (book.getIsbn()==isbn) {**

**books.remove(book);**

**System.*out*.println("book removed");**

**return true;**

**}**

**}**

**System.*out*.println("Book with ISBN"+isbn +"not found");**

**return false;**

**}**

**public void searchbookTitle(String title) {**

**boolean found = false;**

**for (Book book :books) {**

**if(book.getTitle().equalsIgnoreCase(title)) {**

**book.display();**

**found =true;**

**}**

**}**

**if (!found) {**

**System.*out*.println("no book found"+title);**

**}**

**}**

**public void displayallbook() {**

**if (books.isEmpty()) {**

**System.*out*.println("No book in library");**

**return;**

**}**

**System.*out*.println("book in library");**

**for(Book book :books) {**

**book.display();**

**}**

**}**

**}**

**public class LibraryManagement {**

**public static void main(String [] args) {**

**Library lia = new Library();**

**lia.addBook(new Book("harry", "falgun",4555));**

**lia.addBook(new Book("Balbharti","sham",782));**

**lia.addBook(new Book("Mahabharat","Vedvyas",252));**

**lia.displayallbook();**

**lia.searchbookTitle("balbharti");**

**lia.removeBook(4555);**

**lia.displayallbook();**

**}**

**}**

**Question 2.)**

You are developing a Java program to manage an online shopping cart. Implement code to handle the following built-in exceptions:

1. ArrayIndexOutOfBoundsException

2. NumberFormatException

3. ArithmeticException

**import java.util.Scanner;**

**class ShoppingCart {**

**private String[] items;**

**private int itemCount;**

**public ShoppingCart(int capacity) {**

**items = new String[capacity];**

**itemCount = 0;**

**}**

**public void addItem(String item) {**

**try {**

**items[itemCount] = item;**

**itemCount++;**

**System.*out*.println(item + " added to cart.");**

**} catch (ArrayIndexOutOfBoundsException e) {**

**System.*out*.println("cart is full");**

**}**

**}**

**public void removeItem(int index) {**

**try {**

**System.*out*.println("Removing item: " + items[index]);**

**for (int i = index; i < itemCount - 1; i++) {**

**items[i] = items[i + 1];**

**}**

**items[itemCount - 1] = null;**

**itemCount--;**

**} catch (ArrayIndexOutOfBoundsException e) {**

**System.*out*.println("nnvalid");**

**}**

**}**

**public void calculateprice() {**

**Scanner scanner = new Scanner(System.*in*);**

**try {**

**System.*out*.print("Enter total price: ");**

**String totalPriceStr = scanner.nextLine();**

**int totalPrice = Integer.*parseInt*(totalPriceStr);**

**System.*out*.print("Enter number of item ");**

**String itemCountStr = scanner.nextLine();**

**int count = Integer.*parseInt*(itemCountStr);**

**int average = totalPrice / count;**

**System.*out*.println("average price of per " + average);**

**} catch (NumberFormatException e) {**

**System.*out*.println("invaild format");**

**} catch (ArithmeticException e) {**

**System.*out*.println("arithmetic error");**

**}**

**}**

**public void displayCart() {**

**System.*out*.println("items in cart:");**

**for (int i = 0; i < itemCount; i++) {**

**System.*out*.println(i + ": " + items[i]);**

**}**

**}**

**}**

**public class onlineshoping{**

**public static void main(String[] args) {**

**ShoppingCart cart = new ShoppingCart(3);**

**cart.addItem("Laptop");**

**cart.addItem("Mouse");**

**cart.addItem("Keyboard");**

**cart.addItem("Monitor");**

**cart.displayCart();**

**cart.removeItem(5);**

**cart.calculateprice();**

**}**

**}**

**Question 3.)**

You are developing a Java application for managing vehicles in a rental service. Create a class named Vehicle with attributes for make, model, and year. Implement a constructor that initializes these fields. Next, design a subclass named Car with an additional attribute for numberOfDoors. Provide two constructors for the Car class: another that accepts only make and model, chaining to the superclass constructor. Validate the input in each constructor and provide appropriate getter and setter methods to access and modify the attributes of the Vehicle class. Ensure proper usage of inheritance principles.

**class Vehicle {**

**private String make;**

**private String model;**

**private int year;**

**public Vehicle(String make, String model, int year) {**

**setMake(make);**

**this.model =model;**

**this.year=year;**

**}**

**public String getMake() {**

**return make;**

**}**

**public void setMake(String make) {**

**this.make = make;**

**}**

**public String getModel() {**

**return model;**

**}**

**public int getYear() {**

**return year;**

**}**

**}**

**class Car extends Vehicle {**

**private int numberOfDoors;**

**public Car(String make, String model, int year, int numberOfDoors) {**

**super(make, model, year);**

**setNumberOfDoors(numberOfDoors);**

**}**

**public Car(String make, String model) {**

**super(make, model, 0);**

**this.numberOfDoors = 0;**

**}**

**public int getNumberOfDoors() {**

**return numberOfDoors;**

**}**

**public void setNumberOfDoors(int numberOfDoors) {**

**this.numberOfDoors = numberOfDoors;**

**}**

**public void getValue() {**

**System.*out*.println("Make: " + getMake() + " Model: " + getModel() + " Year: " + getYear() + " NumberOfDoors: " + getNumberOfDoors());**

**}**

**}**

**public class VehicleRental{**

**public static void main(String[] args){**

**Car cr=new Car("BMW", "X5", 2025, 5);**

**cr.getValue();**

**Car cr2=new Car("Volvo", "X90",2025,4);**

**cr2.getValue();**

**}**

**}**

**Question 4.)**

Create a Java program to manage an employee database for a company using an ArrayList. Each employee should have attributes such as employee name, employee ID, and department. Implement an Employee class with methods to perform the following operations:

1. Add a new employee to the database.

2. Update an employee’s department using their employee ID.

3. Remove an employee from the database using their employee ID.

4. Display the list of all employees along with their details.

import java.util.\*;

class Employee2 {

private String name;

private int employeeId;

private String department;

public Employee2(String name, int employeeId, String department) {

this.name = name;

this.employeeId = employeeId;

this.department = department;

}

public String getName() {

return name;

}

public int getEmployeeId() {

return employeeId;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

*@Override*

public String toString() {

return "Employee ID: " + employeeId + ", Name: " + name + ", Department: " + department;

}

}

class EmployeeDatabase {

private ArrayList<Employee2> employees;

public EmployeeDatabase() {

employees = new ArrayList<>();

}

public void addEmployee(Employee2 employee) {

employees.add(employee);

System.***out***.println("Employee added successfully.");

}

public void updateEmployeeDepartment(int employeeId, String newDepartment) {

for (Employee2 employee : employees) {

if (employee.getEmployeeId() == employeeId) {

employee.setDepartment(newDepartment);

System.***out***.println("Employee department updated successfully.");

return;

}

}

System.***out***.println("Employee with ID " + employeeId + " not found.");

}

public void removeEmployee(int employeeId) {

Iterator<Employee2> iterator = employees.iterator();

while (iterator.hasNext()) {

Employee2 employee = iterator.next();

if (employee.getEmployeeId() == employeeId) {

iterator.remove();

System.***out***.println("Employee removed successfully.");

return;

}

}

System.***out***.println("Employee with ID " + employeeId + " not found.");

}

public void displayAllEmployees() {

if (employees.isEmpty()) {

System.***out***.println("No employees in the database.");

return;

}

System.***out***.println("--- Employee List ---");

for (Employee2 employee : employees) {

System.***out***.println(employee);

}

System.***out***.println("---------------------");

}

}

public class Employemanagement{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

EmployeeDatabase database = new EmployeeDatabase();

int choice;

do {

System.***out***.println("\n Employee Management System ---");

System.***out***.println("1. Add a new employee");

System.***out***.println("2. Update an employee's department");

System.***out***.println("3. Remove an employee");

System.***out***.println("4. Display all employees");

System.***out***.println("5. Exit");

System.***out***.print("Enter your choice: ");

choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

System.***out***.print("Enter employee name: ");

String name = scanner.nextLine();

System.***out***.print("Enter employee ID: ");

int id = scanner.nextInt();

scanner.nextLine();

System.***out***.print("Enter department: ");

String department = scanner.nextLine();

database.addEmployee(new Employee2(name, id, department));

break;

case 2:

System.***out***.print("Enter employee ID to update: ");

int updateId = scanner.nextInt();

scanner.nextLine();

System.***out***.print("Enter new department: ");

String newDepartment = scanner.nextLine();

database.updateEmployeeDepartment(updateId, newDepartment);

break;

case 3:

System.***out***.print("Enter employee ID to remove: ");

int removeId = scanner.nextInt();

scanner.nextLine();

database.removeEmployee(removeId);

break;

case 4:

database.displayAllEmployees();

break;

case 5:

System.***out***.println("Exiting the program. Goodbye!");

break;

default:

System.***out***.println("Invalid choice. Please try again.");

}

} while (choice != 5);

scanner.close();

}

}

**Question 5.)**

You are developing a Java program to manage user inputs in a ticket booking system. Implement code to handle the following built-in exceptions:

1. InputMismatchException3

2. IllegalArgumentException

3. IndexOutOfBoundsException

import java.util.\*;

public class ticketbooking {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

try {

System.***out***.print("Enter the number of tickets you want to book: ");

int numberOfTickets = scanner.nextInt();

System.***out***.println("You have booked " + numberOfTickets + " tickets.");

} catch (InputMismatchException e) {

System.***out***.println("Error: Please enter a valid number for tickets.");

scanner.next();

}

scanner.close();

}

}

**Question 6.)**

You are creating a Java program to represent different types of employees in a company. Implement a class named Employee with attributes for name, employeeID, and department. Provide a constructor to initialize these fields. Next, create a subclass named Manager with an additional attribute for numberOfTeams managed. The Manager class should have two constructors: one that accepts all fields, and another that accepts only name and employeeID, chaining to the superclass constructor. Include methods to validate input data and implement getter and setter methods for the attributes in the Employee class. Demonstrate the inheritance hierarchy by creating instances of both classes and displaying their details.

import java.util.\*;

public class Employeecompany{

public static void main(String[] args) {

*demonstrateInputMismatchException*();

System.***out***.println();

*demonstrateIllegalArgumentException*();

System.***out***.println();

*demonstrateIndexOutOfBoundsException*();

}

public static void demonstrateInputMismatchException() {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Demonstrating InputMismatchException");

try {

System.***out***.print("Enter a number of tickets: ");

int numberOfTickets = scanner.nextInt();

System.***out***.println("You entered: " + numberOfTickets);

} catch (InputMismatchException e) {

System.***out***.println("Caught InputMismatchException: Please enter a valid number.");

}

}

public static void bookTickets(int numberOfTickets) {

if (numberOfTickets <= 0) {

throw new IllegalArgumentException("Number of tickets must be greater than zero.");

}

System.***out***.println("Successfully booked " + numberOfTickets + " tickets.");

}

public static void demonstrateIllegalArgumentException() {

System.***out***.println("Demonstrating IllegalArgumentException");

try {

*bookTickets*(0);

} catch (IllegalArgumentException e) {

System.***out***.println("Caught IllegalArgumentException:" + e.getMessage());

}

}

public static void demonstrateIndexOutOfBoundsException() {

System.***out***.println("Demonstrating IndexOutOfBoundsException");

List<String> seats = new ArrayList<>();

seats.add("A1");

seats.add("A2");

seats.add("A3");

try {

String selectedSeat = seats.get(5);

System.***out***.println("Selected seat: " + selectedSeat);

} catch (IndexOutOfBoundsException e) {

System.***out***.println("Caught IndexOutOfBoundsException: The requested seat index is out of bounds.");

}

}

}

**Question 7.)**

Develop a Java program for a simple banking system using an ArrayList to manage bank accounts. Each bank account should have attributes including the name of the account holder, account number, and initial balance. Implement the BankAccount class with methods to perform the following actions: (Using array list)

● Allow users to deposit money into their account.

● Allow users to withdraw money from their account if they have sufficient balance. ● Display the current balance of the account.

● Enable users to transfer money from one account to another, provided they have sufficient balance

import java.util.\*;

class BankAccount {

private String accountHolderName;

private int accountNumber;

private double balance;

public BankAccount(String accountHolderName, int accountNumber, double initialBalance) {

this.accountHolderName = accountHolderName;

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

public int getAccountNumber() {

return accountNumber;

}

public double getBalance() {

return balance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.***out***.println("Deposit successful. New balance: $" + String.*format*("%.2f", balance));

} else {

System.***out***.println("Deposit amount must be positive.");

}

}

public boolean withdraw(double amount) {

if (amount > 0) {

if (balance >= amount) {

balance -= amount;

System.***out***.println("Withdrawal successful. New balance: $" + String.*format*("%.2f", balance));

return true;

} else {

System.***out***.println("Insufficient balance.");

return false;

}

} else {

System.***out***.println("Withdrawal amount must be positive.");

return false;

}

}

public void displayBalance() {

System.***out***.println("Current balance for account " + accountNumber + " is $" + String.*format*("%.2f", balance));

}

}

public class bankingsystem {

private static List<BankAccount> *accounts* = new ArrayList<>();

private static Scanner *scanner* = new Scanner(System.***in***);

private static BankAccount findAccount(int accountNumber) {

for (BankAccount account : *accounts*) {

if (account.getAccountNumber() == accountNumber) {

return account;

}

}

return null;

}

public static void main(String[] args) {

*accounts*.add(new BankAccount("falgun dhande", 1001, 500.00));

*accounts*.add(new BankAccount("nikhil nehete", 1002, 1200.50));

*accounts*.add(new BankAccount("lokesh fegade", 1003, 75.25));

int choice;

do {

System.***out***.println("\n banking system menu");

System.***out***.println("1. Deposit");

System.***out***.println("2. Withdraw");

System.***out***.println("3. Transfer Funds");

System.***out***.println("4. Display Balance");

System.***out***.println("5. Exit");

System.***out***.print("Enter your choice: ");

try {

choice = *scanner*.nextInt();

*scanner*.nextLine();

switch (choice) {

case 1:

*handleDeposit*();

break;

case 2:

*handleWithdrawal*();

break;

case 3:

*handleTransfer*();

break;

case 4:

*handleDisplayBalance*();

break;

case 5:

System.***out***.println("Thank you for using our banking system.");

break;

default:

System.***out***.println("Invalid choice. Please try again.");

}

} catch (InputMismatchException e) {

System.***out***.println("Invalid input. Please enter a number.");

*scanner*.nextLine();

choice = 0;

}

} while (choice != 5);

*scanner*.close();

}

private static void handleDeposit() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to deposit: $");

double amount = *scanner*.nextDouble();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.deposit(amount);

} else {

System.***out***.println("Account not found.");

}

}

private static void handleWithdrawal() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to withdraw: $");

double amount = *scanner*.nextDouble();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.withdraw(amount);

} else {

System.***out***.println("Account not found.");

}

}

private static void handleTransfer() {

System.***out***.print("Enter your account number (from): ");

int fromAccountNumber = *scanner*.nextInt();

System.***out***.print("Enter recipient's account number (to): ");

int toAccountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to transfer: $");

double amount = *scanner*.nextDouble();

BankAccount fromAccount = *findAccount*(fromAccountNumber);

BankAccount toAccount = *findAccount*(toAccountNumber);

if (fromAccount == null) {

System.***out***.println("Source account not found.");

return;

}

if (toAccount == null) {

System.***out***.println("Destination account not found.");

return;

}

if (fromAccount.withdraw(amount)) {

toAccount.deposit(amount);

System.***out***.println("Transfer successful.");

}

}

private static void handleDisplayBalance() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.displayBalance();

} else {

System.***out***.println("Account not found.");

}

}

}

. **Question 8.)**

You're developing error handling for a Java program that manages payments in an e-commerce application. Write code to handle the following three, specific payment gateway errors:

● Timeout error: Implement exception handling to catch situations where the payment gateway response times out.

● Invalid card details: Handle exceptions arising from attempts to process payments with invalid card information.

● Insufficient funds: Implement exception handling to manage cases where users attempt to make payments without sufficient funds in their account.

import java.util.\*;

class BankAccount {

private String accountHolderName;

private int accountNumber;

private double balance;

public BankAccount(String accountHolderName, int accountNumber, double initialBalance) {

this.accountHolderName = accountHolderName;

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

public int getAccountNumber() {

return accountNumber;

}

public double getBalance() {

return balance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.***out***.println("Deposit successful. New balance: $" + String.*format*("%.2f", balance));

} else {

System.***out***.println("Deposit amount must be positive.");

}

}

public boolean withdraw(double amount) {

if (amount > 0) {

if (balance >= amount) {

balance -= amount;

System.***out***.println("Withdrawal successful. New balance: $" + String.*format*("%.2f", balance));

return true;

} else {

System.***out***.println("Insufficient balance.");

return false;

}

} else {

System.***out***.println("Withdrawal amount must be positive.");

return false;

}

}

public void displayBalance() {

System.***out***.println("Current balance for account " + accountNumber + " is $" + String.*format*("%.2f", balance));

}

}

public class bankingsystem {

private static List<BankAccount> *accounts* = new ArrayList<>();

private static Scanner *scanner* = new Scanner(System.***in***);

private static BankAccount findAccount(int accountNumber) {

for (BankAccount account : *accounts*) {

if (account.getAccountNumber() == accountNumber) {

return account;

}

}

return null;

}

public static void main(String[] args) {

*accounts*.add(new BankAccount("falgun dhande", 1001, 500.00));

*accounts*.add(new BankAccount("nikhil nehete", 1002, 1200.50));

*accounts*.add(new BankAccount("lokesh fegade", 1003, 75.25));

int choice;

do {

System.***out***.println("\n banking system menu");

System.***out***.println("1. Deposit");

System.***out***.println("2. Withdraw");

System.***out***.println("3. Transfer Funds");

System.***out***.println("4. Display Balance");

System.***out***.println("5. Exit");

System.***out***.print("Enter your choice: ");

try {

choice = *scanner*.nextInt();

*scanner*.nextLine();

switch (choice) {

case 1:

*handleDeposit*();

break;

case 2:

*handleWithdrawal*();

break;

case 3:

*handleTransfer*();

break;

case 4:

*handleDisplayBalance*();

break;

case 5:

System.***out***.println("Thank you for using our banking system.");

break;

default:

System.***out***.println("Invalid choice. Please try again.");

}

} catch (InputMismatchException e) {

System.***out***.println("Invalid input. Please enter a number.");

*scanner*.nextLine();

choice = 0;

}

} while (choice != 5);

*scanner*.close();

}

private static void handleDeposit() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to deposit: $");

double amount = *scanner*.nextDouble();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.deposit(amount);

} else {

System.***out***.println("Account not found.");

}

}

private static void handleWithdrawal() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to withdraw: $");

double amount = *scanner*.nextDouble();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.withdraw(amount);

} else {

System.***out***.println("Account not found.");

}

}

private static void handleTransfer() {

System.***out***.print("Enter your account number (from): ");

int fromAccountNumber = *scanner*.nextInt();

System.***out***.print("Enter recipient's account number (to): ");

int toAccountNumber = *scanner*.nextInt();

System.***out***.print("Enter amount to transfer: $");

double amount = *scanner*.nextDouble();

BankAccount fromAccount = *findAccount*(fromAccountNumber);

BankAccount toAccount = *findAccount*(toAccountNumber);

if (fromAccount == null) {

System.***out***.println("Source account not found.");

return;

}

if (toAccount == null) {

System.***out***.println("Destination account not found.");

return;

}

if (fromAccount.withdraw(amount)) {

toAccount.deposit(amount);

System.***out***.println("Transfer successful.");

}

}

private static void handleDisplayBalance() {

System.***out***.print("Enter account number: ");

int accountNumber = *scanner*.nextInt();

BankAccount account = *findAccount*(accountNumber);

if (account != null) {

account.displayBalance();

} else {

System.***out***.println("Account not found.");

}

}

}

**Question 9.)**

You're developing a Java program to manage individuals within an educational institution. Create a class named "Person" with attributes for the name and age of an individual. Implement a constructor that accepts both fields as arguments. Next, design a subclass named "Student" with an additional attribute for the grade level. Provide two constructors for the Student class: one that accepts all fields including name, age, and grade level, and another that accepts only the name and age, chaining to the superclass constructor. Ensure that your program demonstrates proper inheritance principles, and validate the constructors to ensure valid data is provided during object instantiation,also implement appropriate getter and setter methods to access and modify the attributes of Person class.

import java.util.Objects;

class Person {

private String name;

private int age;

public Person(String name, int age) {

if (name == null || name.trim().isEmpty()) {

throw new IllegalArgumentException("Name cannot be null or empty.");

}

if (age <= 0) {

throw new IllegalArgumentException("Age must be a positive number.");

}

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

if (name == null || name.trim().isEmpty()) {

System.***out***.println("Invalid name provided. Name not updated.");

return;

}

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age <= 0) {

System.***out***.println("Invalid age provided. Age must be a positive number.");

return;

}

this.age = age;

}

*@Override*

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

}

class Student extends Person {

private int gradeLevel;

public Student(String name, int age, int gradeLevel) {

super(name, age);

if (gradeLevel <= 0) {

throw new IllegalArgumentException("Grade level must be a positive number.");

}

this.gradeLevel = gradeLevel;

}

public Student(String name, int age) {

super(name, age);

this.gradeLevel = 0;

}

public int getGradeLevel() {

return gradeLevel;

}

public void setGradeLevel(int gradeLevel) {

if (gradeLevel <= 0) {

System.***out***.println("Invalid grade level provided. Grade level not updated.");

return;

}

this.gradeLevel = gradeLevel;

}

*@Override*

public String toString() {

return "Student [name=" + getName() + ", age=" + getAge() + ", gradeLevel=" + gradeLevel + "]";

}

}

public class Educationalinstitution {

public static void main(String[] args) {

System.***out***.println("--- Demonstrating Person and Student Classes ---");

System.***out***.println("\nCreating a Person instance:");

Person person1 = new Person("Falgun", 22);

System.***out***.println(person1);

person1.setAge(36);

System.***out***.println("Updated age: " + person1);

System.***out***.println("\nCreating a Student instance (all fields):");

Student student1 = new Student("Jane Smith", 20, 12);

System.***out***.println(student1);

student1.setGradeLevel(13);

System.***out***.println("Updated grade level: " + student1);

System.***out***.println("\nCreating a Student instance (chained constructor):");

Student student2 = new Student("Mike Johnson", 21);

System.***out***.println(student2);

student2.setGradeLevel(1);

System.***out***.println("Updated grade level: " + student2);

System.***out***.println("\nDemonstrating constructor validation (invalid age):");

try {

new Person("Invalid Person", 0);

} catch (IllegalArgumentException e) {

System.***out***.println("Caught expected exception: " + e.getMessage());

}

System.***out***.println("\nDemonstrating constructor validation (invalid grade level):");

try {

new Student("Invalid Student", 18, -1);

} catch (IllegalArgumentException e) {

System.***out***.println("Caught expected exception: " + e.getMessage());

}

}

}

**Question 10)**

Create a class Student with:

● int studentId, String name, double grade.

● A constructor to initialize these fields.

● Methods:

○ updateGrade(double newGrade): Updates the grade, but should not accept negative values (handle using exception handling).

○ display(): Prints student details.

import java.util.\*;

class Person {

private String name;

private int age;

public Person(String name, int age) {

if (name == null || name.trim().isEmpty()) {

throw new IllegalArgumentException("Name cannot be null or empty.");

}

if (age <= 0) {

throw new IllegalArgumentException("Age must be a positive number.");

}

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

if (name == null || name.trim().isEmpty()) {

System.***out***.println("Invalid name provided. Name not updated.");

return;

}

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age <= 0) {

System.***out***.println("Invalid age provided. Age must be a positive number.");

return;

}

this.age = age;

}

*@Override*

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

}

class Student extends Person {

private int gradeLevel;

public Student(String name, int age, int gradeLevel) {

super(name, age);

if (gradeLevel <= 0) {

throw new IllegalArgumentException("Grade level must be a positive number.");

}

this.gradeLevel = gradeLevel;

}

public Student(String name, int age) {

super(name, age);

this.gradeLevel = 0;

}

public int getGradeLevel() {

return gradeLevel;

}

public void setGradeLevel(int gradeLevel) {

if (gradeLevel <= 0) {

System.***out***.println("Invalid grade level provided. Grade level not updated.");

return;

}

this.gradeLevel = gradeLevel;

}

*@Override*

public String toString() {

return "Student [name=" + getName() + ", age=" + getAge() + ", gradeLevel=" + gradeLevel + "]";

}

}

public class Educationalinstitution {

public static void main(String[] args) {

System.***out***.println(" checked out the person and student classes");

System.***out***.println("\nmaking a new person:");

Person person1 = new Person("Falgun", 22);

System.***out***.println(person1);

person1.setAge(36);

System.***out***.println("whoa, falgun got older: " + person1);

System.***out***.println("\nmaking a new student, giving all the info:");

Student student1 = new Student("jane smith", 20, 12);

System.***out***.println(student1);

student1.setGradeLevel(13);

System.***out***.println("updated jane's grade level: " + student1);

try {

new Person("Invalid Person", 0);

} catch (IllegalArgumentException e) {

System.***out***.println("caught the problem: " + e.getMessage());

}

System.***out***.println("\nshowin' what happens if u give a bad grade level:");

try {

new Student("Invalid Student", 18, -1);

} catch (IllegalArgumentException e) {

System.***out***.println("caught the problem: " + e.getMessage());

}

}

}

**Question 11)**

Write a Java program that reads a string from the user and attempts to convert it to an integer using Integer.parseInt(). If the input is not a valid integer, handle the NumberFormatException. Additionally, handle NullPointerException if the input is null. Use a finally block to print "Conversion attempt completed."

import java.util.Scanner;

public class nullpointer {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

try {

System.***out***.print("give me a number ");

String input = scanner.nextLine();

int number = Integer.*parseInt*(input);

System.***out***.println("hey, nice! you entered the number " + number);

} catch (NumberFormatException e) {

System.***out***.println("oops, that's not a real number. try again with a numbr.");

} finally {

System.***out***.println("all done with that try.");

}

System.***out***.println("\n");

String nullInput = null;

try {

int number = Integer.*parseInt*(nullInput);

System.***out***.println("whoa, that worked? no way. you got " + number);

} catch (NumberFormatException e) {

System.***out***.println("hey, that's not a number.");

} catch (NullPointerException e) {

System.***out***.println("oh, wait. the string was empty. it's nothin'!");

} finally {

System.***out***.println("all done with that try.");

}

scanner.close();

}

}

**Question 12)**

Patient Management System

You are managing a patient database for a hospital. Each patient has a unique patient ID, a name, a diagnosis, and the number of days admitted. You need to implement a solution using appropriate Java collection classes to efficiently perform the following operations: a. Add a new patient to the database.

b. Remove a patient from the database.

c. Find all patients with a specific diagnosis.

d. Find all patients admitted for more than a given number of days.

import java.util.\*;

class Patient {

private int patientId;

private String name;

private String diagnosis;

private int daysAdmitted;

public Patient(int patientId, String name, String diagnosis, int daysAdmitted) {

this.patientId = patientId;

this.name = name;

this.diagnosis = diagnosis;

this.daysAdmitted = daysAdmitted;

}

public int getPatientId() {

return patientId;

}

public String getDiagnosis() {

return diagnosis;

}

public int getDaysAdmitted() {

return daysAdmitted;

}

*@Override*

public String toString() {

return "patient id: " + patientId + ", name: " + name + ", diagnosis: " + diagnosis + ", days admitted: " + daysAdmitted;

}

}

class Hospital {

private List<Patient> patients;

public Hospital() {

this.patients = new ArrayList<>();

}

public void addPatient(Patient patient) {

patients.add(patient);

System.***out***.println("hey, " + patient.getPatientId() + " is in the system now.");

}

public void removePatient(int patientId) {

Iterator<Patient> iterator = patients.iterator();

while (iterator.hasNext()) {

Patient patient = iterator.next();

if (patient.getPatientId() == patientId) {

iterator.remove();

System.***out***.println("patient " + patientId + " is no longer in the system.");

return;

}

}

System.***out***.println("sorry, cant find patient " + patientId + ".");

}

public List<Patient> findPatientsByDiagnosis(String diagnosis) {

List<Patient> foundPatients = new ArrayList<>();

for (Patient patient : patients) {

if (patient.getDiagnosis().equalsIgnoreCase(diagnosis)) {

foundPatients.add(patient);

}

}

return foundPatients;

}

public List<Patient> findPatientsByDays(int minDays) {

List<Patient> foundPatients = new ArrayList<>();

for (Patient patient : patients) {

if (patient.getDaysAdmitted() > minDays) {

foundPatients.add(patient);

}

}

return foundPatients;

}

}

public class patientmanagement {

public static void main(String[] args) {

Hospital hospital = new Hospital();

System.***out***.println("ok, let's get some patients in here.");

hospital.addPatient(new Patient(101, "falgun", "fever", 3));

hospital.addPatient(new Patient(102, "ram ", "pneumonia", 8));

hospital.addPatient(new Patient(103, "sham", "fever", 2));

hospital.addPatient(new Patient(104, "ramlal", "broken leg", 10));

hospital.addPatient(new Patient(105, "modi", "pneumonia", 5));

System.***out***.println("\n looking' for a fever ");

List<Patient> feverPatients = hospital.findPatientsByDiagnosis("fever");

if (feverPatients.isEmpty()) {

System.***out***.println("nobody's got a fever right now.");

} else {

System.***out***.println("found these people with a fever:");

for (Patient patient : feverPatients) {

System.***out***.println(patient);

}

}

System.***out***.println("\n--- findin' patients who stayed longer than 5 days ---");

List<Patient> longStayPatients = hospital.findPatientsByDays(5);

if (longStayPatients.isEmpty()) {

System.***out***.println("nobody has been here for more than 5 days.");

} else {

System.***out***.println("these folks been here a while:");

for (Patient patient : longStayPatients) {

System.***out***.println(patient);

}

}

System.***out***.println("\n to remove falgun from the system ");

hospital.removePatient(101);

System.***out***.println("\n to remove a patient here ");

hospital.removePatient(999);

}

}