## Java Advance -Exception & Collection

1. Ramesh is developing a student management system for a university. In this system, you have Student class to represent student information. You are asked to help Ramesh to handle exception which can be occurred into program according to following Scenarios:

* class Student with attributes roll no, name, age and course. Initialize values through parameterized constructor.
* If the age of the student is not between 15 and 21 then generate a user-defined exception "AgeNotWithinRangeException".
* If a name contains numbers or special symbols, raise exception "NameNot ValidException". Define the two exception classes.

public class Student01 {  
 private int rollno;  
 private String name;  
 private int age;  
 private String course;  
 // Constructor to initialize student details and validate inputs  
 public Student01(int rollno, String name, int age, String course) throws AgeNotWithinRangeException, NameNotValidException{  
 if(age<15 || age>21) { // Validate age: should be between 15 and 21  
 throw new AgeNotWithinRangeException("Age must be between 15 and 21.");  
 }  
 if (!name.matches("[a-zA-Z\\s]+")) { // Validate name: should not contain numbers or special characters  
 throw new NameNotValidException("Name must not contain numbers and special characters.");  
 }  
 // Assign values if validation passes  
 this.rollno = rollno;  
 this.name = name;  
 this.age = age;  
 this.course = course;  
 }  
 @Override  
 // Overriding toString() method  
 public String toString() {  
 return "Roll No: "+ rollno+" Name: "+name+" Age: "+age+" Course: "+course;  
 }  
}

// Custom exception for invalid name  
public class NameNotValidException extends Exception {  
 // Constructor to pass the error message to the Exception superclass  
 public NameNotValidException(String message){  
 super(message);  
 }  
}

// Custom exception for invalid age  
public class AgeNotWithinRangeException extends Exception{  
 // Constructor to pass the error message to the Exception superclass  
 public AgeNotWithinRangeException(String message){  
 super(message);  
  
 }  
}

import java.util.Scanner;  
  
public class Test01 {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*); // Scanner object to take user input  
 System.*out*.print("Number of students: ");  
 int n=sc.nextInt(); // Number of students to register  
 Student01[] students=new Student01[n];  
 System.*out*.println("Enter Students details");  
 try {  
 for (int i = 0; i < n; i++) {  
 // Loop to collect details for each student  
 System.*out*.print("Enter Rollno.: ");  
 int rollno=sc.nextInt();  
 sc.nextLine();  
 System.*out*.print("Enter Name: ");  
 String name= sc.nextLine();  
 System.*out*.print("Enter Age: ");  
 int age=sc.nextInt();  
 sc.nextLine();  
 System.*out*.print("Enter course: ");  
 String course=sc.nextLine();  
 // Create a Student01 object and add it to the array  
 students[i]=new Student01(rollno, name, age, course);  
 }  
 } catch (AgeNotWithinRangeException e) {  
 // Handle age-related exceptions  
 System.*out*.println("Error: "+e.getMessage());;  
 } catch (NameNotValidException e) {  
 // Handle name validation exceptions  
 System.*out*.println("Error: "+e.getMessage());  
 }  
 System.*out*.println();  
 for (int i = 0; i < n; i++) {

if (students[i] != null) {  
 // Display details of all registered students  
 System.*out*.println(students[i].toString());

}  
 }  
  
  
 }  
}

Output:

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

2. Create a class Voter(voterId, name, age) with parameterized constructor. The parameterized constructor should throw a checked exception if age is less than 18. The message of exception is "invalid age for voter ".

public class Voter02 {  
 private int voterID;  
 private String name;  
 private int age;  
 // Constructor to initialize Voter02 object and validate age  
 public Voter02(int voterID, String name, int age) throws InvalidAgeException02{  
 if(age<18){ // Check if age is less than 18 and throw custom exception if true  
 throw new InvalidAgeException02("Invalid Age for voting");  
 }  
 this.voterID=voterID;  
 this.name=name;  
 this.age=age;  
 }  
 // Method to display voter details  
 public void display(){  
 System.*out*.println("VoterID: "+voterID+" Name: "+name+" Age: "+age);  
 }  
}

// Custom checked exception for invalid voter age  
public class InvalidAgeException02 extends Exception {  
 // Constructor to initialize the exception with a custom message  
 public InvalidAgeException02(String message) {  
 super(message); // Pass the message to the Exception superclass  
 }  
  
}

import java.util.Scanner;  
  
public class Test02 {  
 public static void main(String[] args) {  
  
 Scanner sc=new Scanner(System.*in*); // Scanner object to take user input  
 System.*out*.print("Enter no. of Voters: ");  
 int n=sc.nextInt(); // Number of voters to be registered  
 Voter02[] voters=new Voter02[n]; // Array to store voter objects  
 int count=0;  
 try {  
 // Loop to get voter details and create voter objects  
 for (int i = 0; i < n; i++) {  
 System.*out*.print("Enter voterID: ");  
 int voterID = sc.nextInt();  
 sc.nextLine();  
 System.*out*.print("Enter name: ");  
 String name = sc.nextLine();  
 System.*out*.print("Enter age: ");  
 int age = sc.nextInt();  
 voters[i] = new Voter02(voterID, name, age); // Create Voter02 object and add to array  
 count++;  
 System.*out*.println("Voter added Successfully");  
 }  
 } catch (InvalidAgeException02 e) {  
 // Handle the custom exception for invalid voter age  
 System.*out*.println("Error: "+e.getMessage());;  
 }  
 System.*out*.println();  
 // Display registered voters  
 System.*out*.println("Registered voters list:");  
 for (int i = 0; i < n; i++) {  
 if(count==0){  
 System.*out*.println(0);  
 return;  
 }else{  
 voters[i].display();  
 }  
 }  
 }  
}

Output:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

3. Store name of weekdays in an array (starting from "Sunday" at 0 index). Ask day position from user and print day name. Handle array index out of bound exception and give proper message if user enters day index outside range (0-6).

import java.util.Scanner;  
  
public class Test03 {  
 public static void main(String[] args) {  
 // Scanner object to get input from the user  
 Scanner sc= new Scanner(System.*in*);  
 // Array to store names of the weekdays  
 String[] weekDays={"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"};  
 try {  
 System.*out*.print("Enter day index (0-6): ");  
 int index = sc.nextInt();  
 // Print corresponding weekday based on the index  
 System.*out*.println(weekDays[index]);  
 } catch (ArrayIndexOutOfBoundsException e) {  
 // Handle cases when the user enters an invalid index  
 System.*out*.println("Error: Index out of range. Day index must be between 0 to 6 ");  
 }  
 }  
}

Output:

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

4.Create a HashMap where keys are student names (strings) and values are their corresponding grades (integers). Create methods to add a new student, remove a student, and Display up a student's grade by name.

import java.util.HashMap;  
import java.util.Scanner;  
  
public class Test04 {  
 // HashMap to store student names(keys) and grades(values)  
 public static HashMap<String,Integer> *students*=new HashMap<>();  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 //addStudent("adam",92);  
 // Menu to interact with user  
 System.*out*.println("1. Add students and grades");  
 System.*out*.println("2. Remove student");  
 System.*out*.println("3. Display ");  
 System.*out*.println("4. Exit");  
  
 while(true) {  
 System.*out*.print("Enter choice: ");  
 int choice = sc.nextInt();  
 sc.nextLine();  
 switch (choice) {  
 case 1:  
 // Adding a new student  
 System.*out*.print("Enter student name: ");  
 String name = sc.nextLine();  
 System.*out*.print("Enter grade: ");  
 int grade = sc.nextInt();  
 sc.nextLine();  
 *addStudent*(name, grade);  
 break;  
 case 2:  
 // Removing an existing student  
 System.*out*.println("Enter student name to remove: ");  
 String removeName = sc.nextLine();  
 *removeStudent*(removeName);  
 break;  
 case 3:  
 // Displaying student's grade  
 System.*out*.println("Enter student name to display grade: ");  
 String displayName = sc.nextLine();  
 *displayStudent*(displayName);  
 break;  
 case 4:  
 // Exiting the program  
 System.*out*.println("Exit");  
 return;  
 default:  
 // Handling invalid choice  
 System.*out*.println("Invalid choice");  
 }  
 }  
 }  
 // Method to add a student  
 public static void addStudent(String name,int grade){  
 *students*.put(name,grade);  
 }  
 // Method to remove a student  
 public static void removeStudent(String name){  
 if(*students*.containsKey(name)){  
 *students*.remove(name);  
 System.*out*.println("removed successfully");  
 }else{  
 System.*out*.println("Name not found");  
 }  
 }  
 // Method to display specific student's grade  
 public static void displayStudent(String name){  
 if(*students*.containsKey(name)){  
 System.*out*.println(name+"'s grade:"+*students*.get(name));  
 }else{  
 System.*out*.println("Name not found");  
 }  
 }  
}

Output:

A screenshot of a computer program

Description automatically generated

5. Use Collection Classes to store Integers Create some methods for following functionalities.

a. Include functions for pushing elements onto the stack.

b. popping elements from the stack.

c. checking if the stack is empty.

import java.util.Scanner;  
import java.util.Stack;  
  
public class Test05 {  
 // Stack to hold integers  
 private static Stack<Integer> *stack*=new Stack<>();  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 // Display menu options  
 System.*out*.println("1. Add integer to stack.");  
 System.*out*.println("2. Delete an integer from stack.");  
 System.*out*.println("3. Is stack empty? ");  
 System.*out*.println("4. Display elements in stack.");  
 System.*out*.println("5. Exit");  
 // loop runs until the user chooses to exit  
 while(true){  
 System.*out*.print("Enter choice: ");  
 int choice= sc.nextInt();  
 // Switch case to handle user input  
 switch(choice){  
 case 1:  
 // Add an integer to the stack  
 System.*out*.print("Enter integer to add: ");  
 int n=sc.nextInt();  
 *add*(n);  
 break;  
 case 2:  
 // Delete the top element from the stack  
 *delete*();  
 break;  
 case 3:  
 // Check if the stack is empty  
 System.*out*.println("Is stack empty? "+*isEmpty*());  
 break;  
 case 4:  
 // Display all elements in the stack  
 *display*();  
 break;  
 case 5:  
 // Exit the program  
 System.*out*.print("Exit");  
 return;  
 default:  
 // Handle invalid input  
 System.*out*.println("Invalid choice");  
 }  
 }  
 }  
 // Method to add an integer  
 public static void add(Integer n){  
 *stack*.add(n);  
 System.*out*.println(n+" is added to stack");  
 }  
 // Method to delete the top integer  
 public static void delete(){  
 *stack*.pop();  
 System.*out*.println("Top element in stack removed succesfully");  
 }  
 // Method to check if the stack is empty  
 public static boolean isEmpty(){  
 return *stack*.isEmpty();  
 }  
 // Method to display all elements  
 public static void display(){  
 System.*out*.println(*stack*);  
 }  
  
}

Output:

A screenshot of a computer program

Description automatically generated