## **Lab 12: Exception Handling in Java**

## **Object Oriented Programming**

## **Instructor: Dr. Nazia Pervaiz**

## **Lab Engineer: Ms Bareera Anam**

**Class: BSDS -02A**

**AILYA ZAINAB**

**( 5 2 3 5 0 6 )**

**TASK#01**

**INPUT:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

// Custom Exceptions

class AgeRestrictionException extends Exception {

public AgeRestrictionException(String message) {

super(message);

}

}

class CourseFullException extends Exception {

public CourseFullException(String message) {

super(message);

}

}

class DuplicateCourseRegistrationException extends Exception {

public DuplicateCourseRegistrationException(String message) {

super(message);

}

}

class StudentNotFoundException extends Exception {

public StudentNotFoundException(String message) {

super(message);

}

}

// Course Class

class Course {

private String courseId;

private String name;

private int maxSeats;

private int enrolledStudents;

public Course(String courseId, String name, int maxSeats) {

this.courseId = courseId;

this.name = name;

this.maxSeats = maxSeats;

this.enrolledStudents = 0;

}

public String getCourseId() {

return courseId;

}

public String getName() {

return name;

}

public int getMaxSeats() {

return maxSeats;

}

public int getEnrolledStudents() {

return enrolledStudents;

}

public boolean hasAvailableSeats() {

return enrolledStudents < maxSeats;

}

public void enrollStudent() {

enrolledStudents++;

}

@Override

public String toString() {

return name + " (ID: " + courseId + ") - " + enrolledStudents + "/" + maxSeats + " seats";

}

}

// Student Class

class Student {

private String studentId;

private String name;

private int age;

private List<Course> registeredCourses;

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

this.studentId = studentId;

this.name = name;

this.age = age;

this.registeredCourses = new ArrayList<>();

}

public String getStudentId() {

return studentId;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public List<Course> getRegisteredCourses() {

return registeredCourses;

}

public void registerForCourse(Course course) throws AgeRestrictionException,

CourseFullException,

DuplicateCourseRegistrationException {

if (age < 17) {

throw new AgeRestrictionException("Student must be at least 17 years old to register for courses.");

}

if (!course.hasAvailableSeats()) {

throw new CourseFullException("Course " + course.getName() + " is already full.");

}

for (Course registeredCourse : registeredCourses) {

if (registeredCourse.getCourseId().equals(course.getCourseId())) {

throw new DuplicateCourseRegistrationException("Student is already registered for " + course.getName());

}

}

course.enrollStudent();

registeredCourses.add(course);

}

public void displayRegisteredCourses() {

System.out.println("\n" + name + "'s Registered Courses:");

if (registeredCourses.isEmpty()) {

System.out.println("No courses registered yet.");

} else {

for (Course course : registeredCourses) {

System.out.println("- " + course);

}

}

}

}

// Main Application

public class UniversityRegistrationSystem {

private static List<Course> courses = new ArrayList<>();

private static List<Student> students = new ArrayList<>();

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

private static int nextStudentId = 1;

public static void main(String[] args) {

initializeCourses();

while (true) {

System.out.println("\nUniversity Course Registration System");

System.out.println("1. Display Available Courses");

System.out.println("2. Register as New Student");

System.out.println("3. Register for Course");

System.out.println("4. View My Registered Courses");

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

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

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

displayAvailableCourses();

break;

case 2:

registerNewStudent();

break;

case 3:

registerForCourse();

break;

case 4:

viewStudentCourses();

break;

case 5:

System.out.println("Exiting system...");

return;

default:

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

}

}

}

private static void initializeCourses() {

courses.add(new Course("MATH101", "Introduction to Mathematics", 30));

courses.add(new Course("CS101", "Computer Science Fundamentals", 25));

courses.add(new Course("ENG201", "Advanced English Composition", 20));

courses.add(new Course("PHY101", "Physics Basics", 15));

courses.add(new Course("CHEM101", "General Chemistry", 20));

}

private static void displayAvailableCourses() {

System.out.println("\nAvailable Courses:");

for (int i = 0; i < courses.size(); i++) {

System.out.println((i + 1) + ". " + courses.get(i));

}

}

private static void registerNewStudent() {

System.out.println("\nStudent Registration");

System.out.print("Enter your name: ");

String name = scanner.nextLine();

System.out.print("Enter your age: ");

int age = scanner.nextInt();

scanner.nextLine(); // Consume newline

String studentId = "S" + String.format("%03d", nextStudentId++);

Student newStudent = new Student(studentId, name, age);

students.add(newStudent);

System.out.println("\nRegistration successful!");

System.out.println("Your Student ID: " + studentId);

System.out.println("Please remember your ID for future logins.");

}

private static Student authenticateStudent() throws StudentNotFoundException {

System.out.print("\nEnter your Student ID: ");

String studentId = scanner.nextLine();

for (Student student : students) {

if (student.getStudentId().equalsIgnoreCase(studentId)) {

return student;

}

}

throw new StudentNotFoundException("Student with ID " + studentId + " not found.");

}

private static void registerForCourse() {

try {

Student student = authenticateStudent();

displayAvailableCourses();

System.out.print("Enter course number to register: ");

int courseNum = scanner.nextInt() - 1;

scanner.nextLine(); // Consume newline

if (courseNum < 0 || courseNum >= courses.size()) {

System.out.println("Invalid course selection.");

return;

}

Course course = courses.get(courseNum);

System.out.println("\nAttempting to register " + student.getName() +

" for " + course.getName() + "...");

student.registerForCourse(course);

System.out.println("Registration successful!");

} catch (StudentNotFoundException e) {

System.out.println("Error: " + e.getMessage());

System.out.println("Please register as a new student first.");

} catch (IndexOutOfBoundsException e) {

System.out.println("Invalid course selection.");

} catch (AgeRestrictionException e) {

System.out.println("Registration failed: " + e.getMessage());

} catch (CourseFullException e) {

System.out.println("Registration failed: " + e.getMessage());

} catch (DuplicateCourseRegistrationException e) {

System.out.println("Registration failed: " + e.getMessage());

} finally {

// This will be handled in the viewStudentCourses method

}

}

private static void viewStudentCourses() {

try {

Student student = authenticateStudent();

student.displayRegisteredCourses();

} catch (StudentNotFoundException e) {

System.out.println("Error: " + e.getMessage());

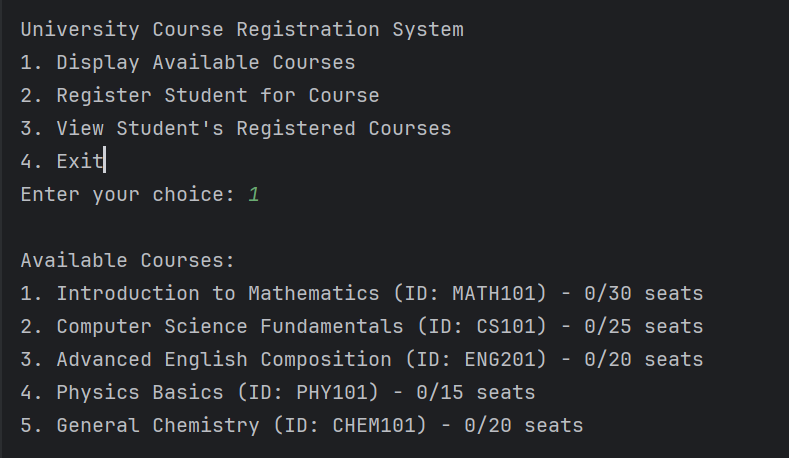
System.out.println("Please register as a new student first.");

}

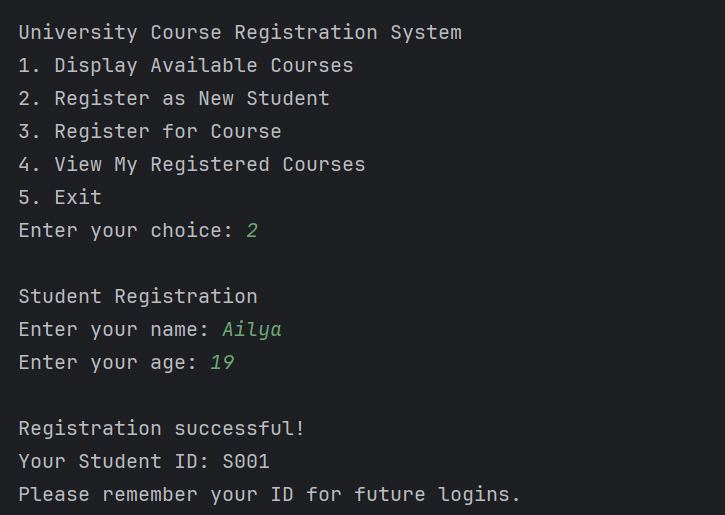
}

}

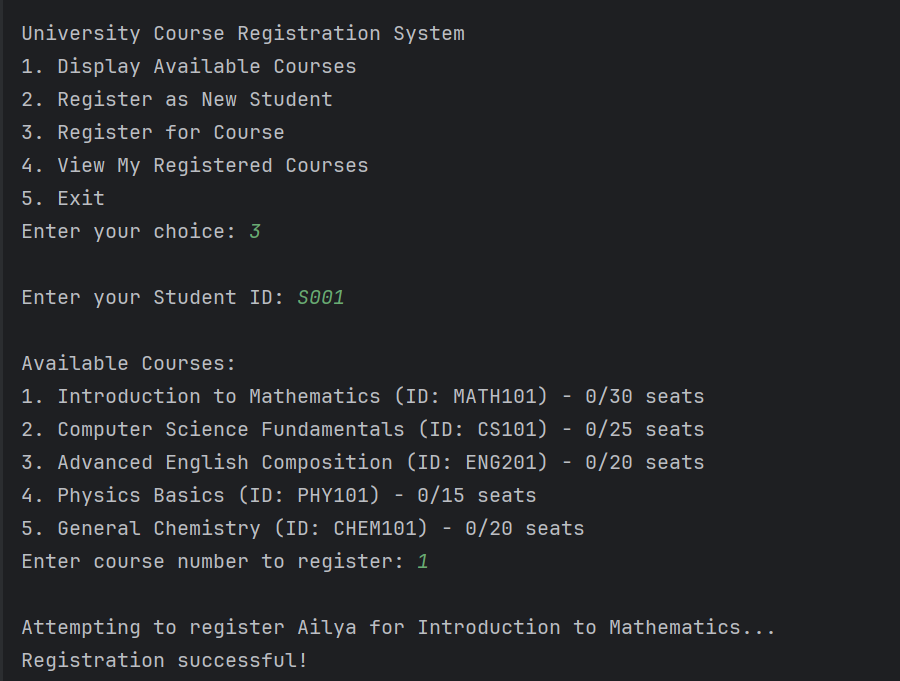
**OUTPUT:**



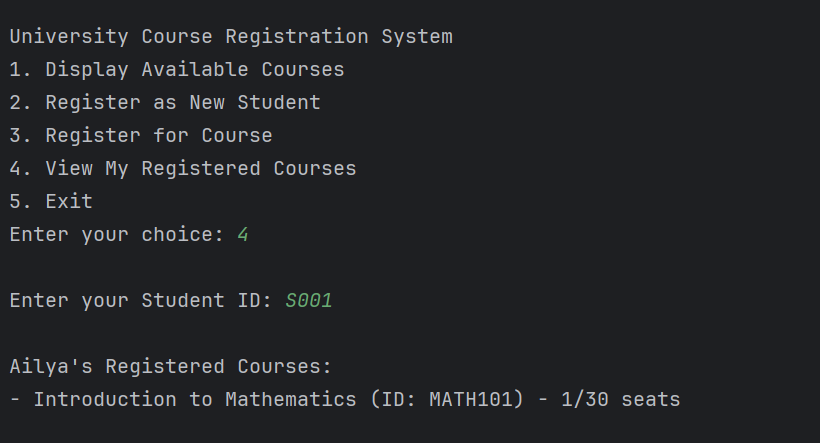
THE LIST OF AVAILABLE COURSES OFFERED AT THE UNIVERSITY



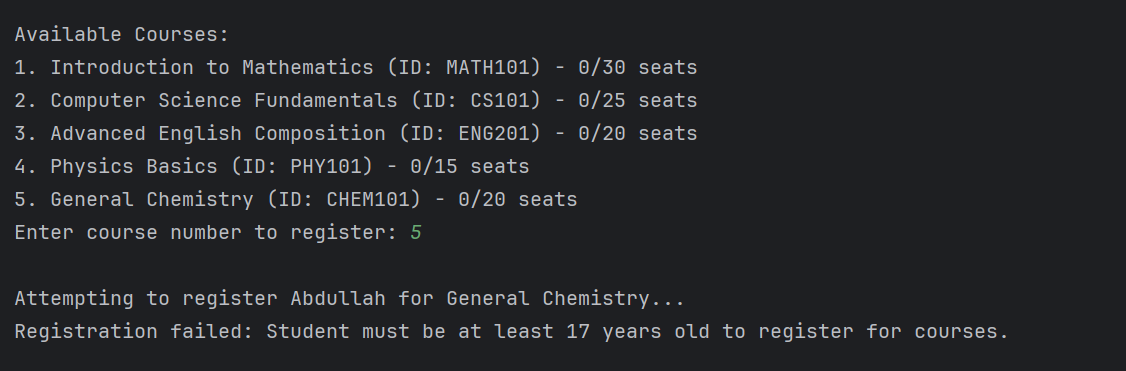
THE LOGIN SYSTEM FOR STUDENTS PROVIDING NAME AND AGE



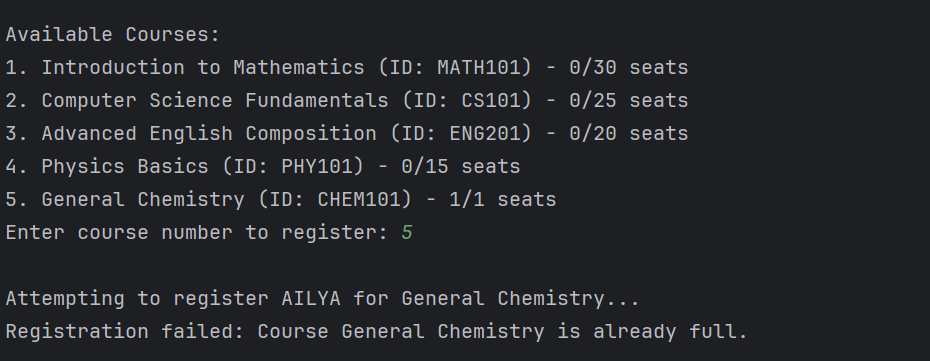
STUDENTLOGGING IN AND REGISTERING FOR COURSES



STUDENTS REGISTERED COURSE AND THE NUMBER OF COURSE SEATS AVAILED BY STUDENTS



STUDENTS UNDER 17 YEARS OF AGE ARE NOT ABLE TO APPLY FOR ANY COURSE



WHEN A STUDENT ATTEMPTS TO APPLY FOR CHEMISTRY 101 COURSE THE REGISTRATION IS CANCELLED AS THE SEATS ARE ALREADY FULL

**TASK #02**

**INPUT:**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

// Custom Exceptions

class InvalidGuestAgeException extends Exception {

public InvalidGuestAgeException(String message) {

super(message);

}

}

class InvalidEmailFormatException extends Exception {

public InvalidEmailFormatException(String message) {

super(message);

}

}

class RoomNotAvailableException extends Exception {

public RoomNotAvailableException(String message) {

super(message);

}

}

class Room {

private String type;

private double price;

private int availableRooms;

public Room(String type, double price, int availableRooms) {

this.type = type;

this.price = price;

this.availableRooms = availableRooms;

}

public String getType() {

return type;

}

public double getPrice() {

return price;

}

public int getAvailableRooms() {

return availableRooms;

}

public void bookRoom() throws RoomNotAvailableException {

if (availableRooms <= 0) {

throw new RoomNotAvailableException("No " + type + " rooms available");

}

availableRooms--;

}

@Override

public String toString() {

return String.format("%-15s $%-10.2f %d available", type, price, availableRooms);

}

}

class Hotel {

private String name;

private Map<String, Room> rooms;

public Hotel(String name) {

this.name = name;

this.rooms = new HashMap<>();

initializeRooms();

}

private void initializeRooms() {

rooms.put("S", new Room("Standard", 100.00, 10));

rooms.put("D", new Room("Deluxe", 150.00, 5));

rooms.put("P", new Room("Premium", 250.00, 3));

rooms.put("SU", new Room("Suite", 400.00, 2));

}

public void displayAvailableRooms() {

System.out.println("\nAvailable Rooms at " + name + ":");

System.out.println("Type Price Availability");

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

for (Room room : rooms.values()) {

System.out.println(room);

}

}

public Room getRoom(String type) throws RoomNotAvailableException {

Room room = rooms.get(type.toUpperCase());

if (room == null) {

throw new RoomNotAvailableException("Invalid room type selected");

}

if (room.getAvailableRooms() <= 0) {

throw new RoomNotAvailableException("No " + room.getType() + " rooms available");

}

return room;

}

}

class Guest {

private String name;

private int age;

private String email;

public Guest(String name, int age, String email) throws InvalidGuestAgeException, InvalidEmailFormatException {

if (age < 18) {

throw new InvalidGuestAgeException("Guest must be at least 18 years old");

}

if (!email.contains("@") || !email.contains(".")) {

throw new InvalidEmailFormatException("Invalid email format. Must contain '@' and '.'");

}

this.name = name;

this.age = age;

this.email = email;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public String getEmail() {

return email;

}

}

public class HotelReservationSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Hotel hotel = new Hotel("Grand Paradise");

while (true) {

System.out.println("\nWelcome to " + hotel.getName() + " Reservation System");

System.out.println("1. View Available Rooms");

System.out.println("2. Make a Reservation");

System.out.println("3. Exit");

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

int choice;

try {

choice = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {

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

continue;

}

switch (choice) {

case 1:

hotel.displayAvailableRooms();

break;

case 2:

makeReservation(hotel, scanner);

break;

case 3:

System.out.println("Thank you for using our reservation system. Goodbye!");

scanner.close();

return;

default:

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

}

}

}

private static void makeReservation(Hotel hotel, Scanner scanner) {

boolean bookingSuccess = false;

try {

System.out.println("\nEnter Guest Details:");

System.out.print("Full Name: ");

String name = scanner.nextLine();

System.out.print("Age: ");

int age = Integer.parseInt(scanner.nextLine());

System.out.print("Email: ");

String email = scanner.nextLine();

Guest guest = new Guest(name, age, email);

hotel.displayAvailableRooms();

System.out.print("\nEnter room type (S/D/P/SU): ");

String roomType = scanner.nextLine().toUpperCase();

Room selectedRoom = hotel.getRoom(roomType);

selectedRoom.bookRoom();

System.out.println("\nBooking Successful!");

System.out.println("Guest: " + guest.getName());

System.out.println("Room Type: " + selectedRoom.getType());

System.out.println("Price: $" + selectedRoom.getPrice());

bookingSuccess = true;

} catch (NumberFormatException e) {

System.out.println("Invalid age format. Please enter a number.");

} catch (InvalidGuestAgeException e) {

System.out.println("Error: " + e.getMessage());

} catch (InvalidEmailFormatException e) {

System.out.println("Error: " + e.getMessage());

} catch (RoomNotAvailableException e) {

System.out.println("Error: " + e.getMessage());

} finally {

if (!bookingSuccess) {

System.out.println("Booking failed. Please try again.");

}

}

}

private static String getHotelName() {

return "Grand Paradise Hotel";

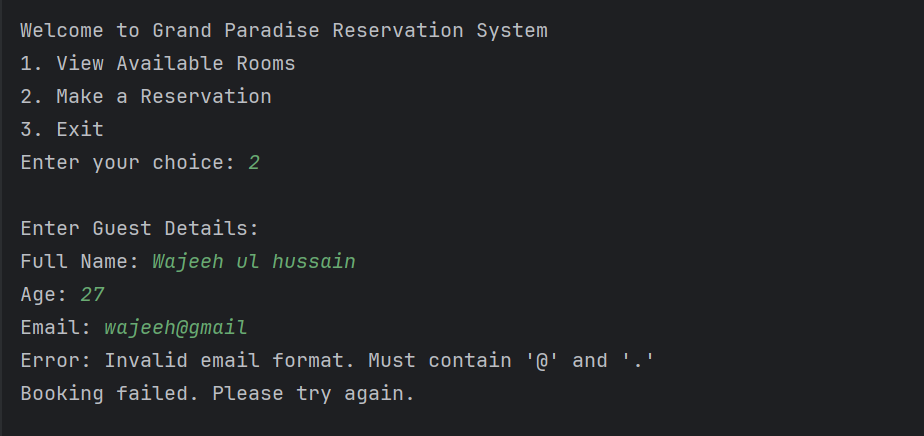
}

}

**OUTPUT:**



AVAILABLE ROOMS AT HOTEL BEING DISPLAYED



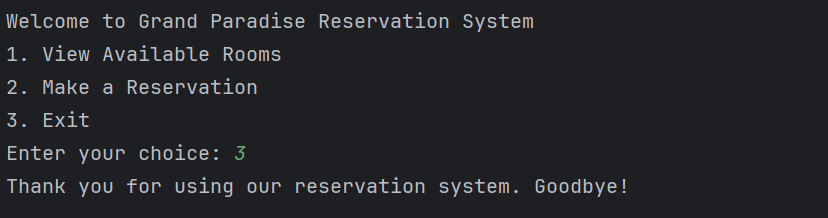
USER TRYING TO PROVIDE CREDENTIALS FOR RESERVATION



USER PROVIDING VALID DATA



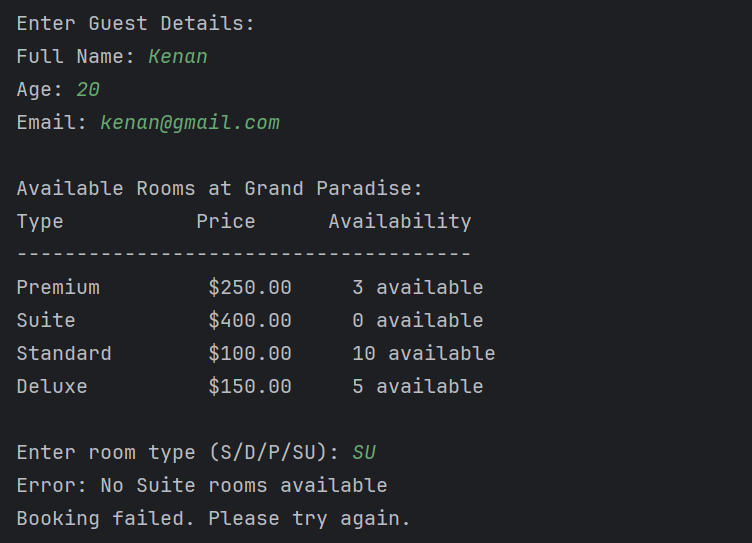
USER SELECTING THE ROOM FOR HOTEL STAY



USER EXITING THE SYSTEM



INVALID AGE EXCEPTION IN THE SYSTEM



USER GETS ERROR WHEN HE TRIES TO BOOK SUITE BECAUSE THE ROOMS ARE ALL TAKEN