**Assignment 1**

**Roll no 248121**

**PRN 240841220016**

**Name: Akshay Chavan Batch B**

**Initial Assignment on exception handling, validation and interface in java.**

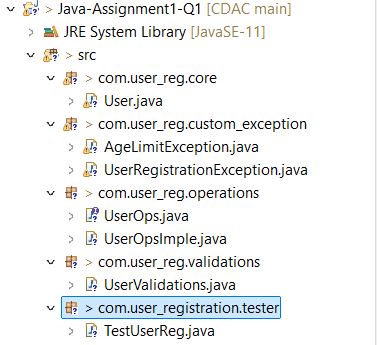
**Q 1.User Registration with Age Limit and login Validation**

**Explanation**

1. **Create User Class: age (int), username and password (string).**
2. **Add suitable constructor and toString.**
3. **Add Custom Exceptions:**
   * **UserRegistrationException handles general registration errors.**
   * **AgeLimitException specifically handles cases where the age does not meet the minimum requirement.**
4. **User Registration Logic:**
   * **The registerUser method validates the username, password, and age.**
   * **If any validation fails, the corresponding exception is thrown.**
5. **Validation Methods:**
   * **validateAge(int age) checks if the age is below the defined minimum (18 in this case) and throws AgeLimitException if it is.**
   * **username and password must be > than 6 characters and should not be empty.**
6. **Tester**
   * **Add valid registration for user. (Attempts with invalid usernames, passwords, and age checks.)**
   * **Display user details.**

**(consider suitable package hierarchy for the same.)**

**Metadata**



**Code**

User:

**package** com.user\_reg.core;

//Core class

**public** **class** User {

**private** **int** userId, age;

**private** String username, password;

**private** **static** **int** *userIdGenerator*;

**static** {

*userIdGenerator* = 100;

}

//constr

**public** User(String username, String password, **int** age) {

**this**.username = username;

**this**.password = password;

**this**.age = age;

**this**.userId = ++*userIdGenerator*;

}

//override toString()

@Override

**public** String toString() {

**return** "UserId " + **this**.userId + " Username " + **this**.username;

}

}

AgeLimitException:

**package** com.user\_reg.custom\_exception;

**public** **class** AgeLimitException **extends** Exception {

**public** AgeLimitException(String message) {

**super**(message);

}

}

UserRegistrationException:

**package** com.user\_reg.custom\_exception;

**public** **class** UserRegistrationException **extends** Exception {

**public** UserRegistrationException(String message) {

//calls Exception's constr -> which calls Throwable's constr which sets the mesg

**super**(message);

}

}

UserOps:

package com.user\_reg.operations;

import com.user\_reg.custom\_exception.AgeLimitException;

import com.user\_reg.custom\_exception.UserRegistrationException;

//has methods for User regs

public interface UserOps {

//since is an interface, has public abstract by default

void registerUser(String username, String password, int age) throws UserRegistrationException, AgeLimitException;

void dispalyAllUsers();

}

UserOpsImple:

package com.user\_reg.operations;

import java.util.ArrayList;

import java.util.List;

import com.user\_reg.core.User;

import com.user\_reg.custom\_exception.AgeLimitException;

import com.user\_reg.custom\_exception.UserRegistrationException;

import static com.user\_reg.validations.UserValidations.\*;

public class UserOpsImple implements UserOps {

//Ref arrayList to store all users

List<User> users;

public UserOpsImple() {

users = new ArrayList<User>();

}

@Override

public void registerUser(String username, String password, int age) throws UserRegistrationException, AgeLimitException {

//check if username already exists first and if user and age valid

validateUser(username, password);

//check if age is valid

validateAge(age);

//if all well create user

users.add(new User(username, password, age));

System.out.println("User " + username + " registered successfully!");

}

@Override

public void dispalyAllUsers() {

if (users.isEmpty()) {

System.out.println("No users registered yet!");

return;

}

for (User u : users) {

if (u != null) {

System.out.println(u);

}

}

}

}

UserValidations:

package com.user\_reg.validations;

import com.user\_reg.custom\_exception.AgeLimitException;

import com.user\_reg.custom\_exception.UserRegistrationException;

public class UserValidations {

public static final int MIN\_AGE = 18;

public static void validateUser(String username, String password) throws UserRegistrationException{

if (username.length() <= 6) {

throw new UserRegistrationException("Username should be > 6 char length!!");

}

if (password.length() <= 6) {

throw new UserRegistrationException("Passoword should be > 6 char length!!");

}

}

public static void validateAge(int age) throws AgeLimitException{

if (age < MIN\_AGE)

throw new AgeLimitException("Not eligible for reg! User must be older than 18 years ;(");

}

}

Tester:

package com.user\_registration.tester;

import java.util.Scanner;

import com.user\_reg.operations.UserOps;

import com.user\_reg.operations.UserOpsImple;

//Tester class for User regs

public class TestUserReg {

public static void main(String[] args) {

try(Scanner sc = new Scanner(System.in)){

UserOps ops = new UserOpsImple(); //create instance of operations

boolean exit = false;

while (!exit) {

System.out.println("User Registrations");

System.out.println("1. Register user\n"

+ "2. Display all users\n");

System.out.println("Enter choice:");

try {

switch (sc.nextInt()) {

case 1:

//Reg User

System.out.println("Enter user details as username, password, age");

ops.registerUser(sc.next(), sc.next(), sc.nextInt());

break;

case 2:

ops.dispalyAllUsers();

break;

case 0:

exit = true; //exit the loop

break;

}

} catch (Exception e) {

//catches all exceps

System.out.println(e);

sc.nextLine();

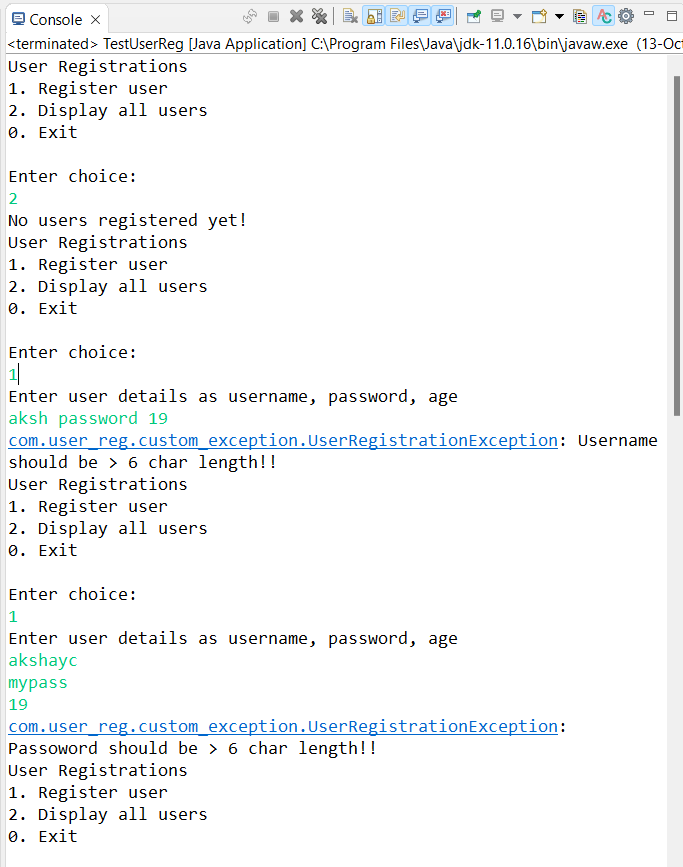
}

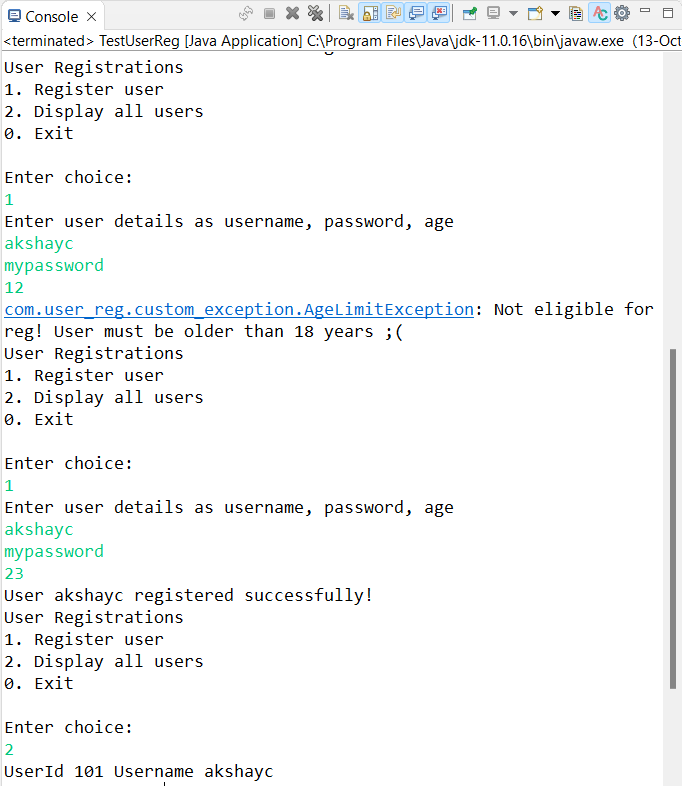
}

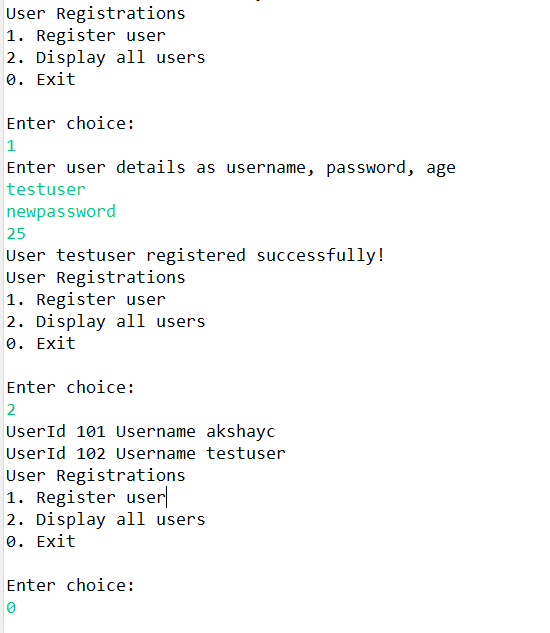
}

}

}

**Output  
**

****

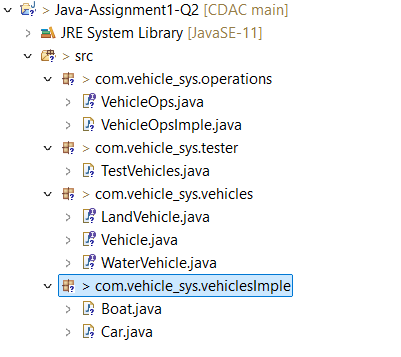


**Q 2. Interface Hierarchy for a Vehicle System**

**Explanation**

1. **Base Interface (Vehicle):**
   * **create a base interface vehicle including basic methods start() and stop().**
2. **Extended Interfaces:**
   * **LandVehicle: Extends Vehicle and adds the method drive(), which is specific to vehicles that operate on land.**
   * **WaterVehicle: Extends Vehicle and adds the method sail(), specific to vehicles that operate on water.**
3. **Concrete Classes:**
   * **Car: Implements the LandVehicle interface, providing implementations for all methods.**
   * **Boat: Implements the WaterVehicle interface, providing its own method implementations.**
4. **Tester (VehicleTest):**
   * **It creates instances of Car and Boat invoking their methods to show the functionality.**

**Metadata**



**Code**

Vehicle:

**package** com.vehicle\_sys.vehicles;

//Base interface

**public** **interface** Vehicle {

**public** **abstract** **void** start();

**void** stop();//public abstract keywords are added implicitely by javac.

}

LandVehicle:

**package** com.vehicle\_sys.vehicles;

**public** **interface** LandVehicle **extends** Vehicle {

**void** drive();

}

WaterVehicle:

**package** com.vehicle\_sys.vehicles;

**public** **interface** WaterVehicle **extends** Vehicle {

**void** sail();

}

Car:

**package** com.vehicle\_sys.vehiclesImple;

**import** com.vehicle\_sys.vehicles.LandVehicle;

**public** **class** Car **implements** LandVehicle {

//no data members added since learning interface hierrachy for now.

**public** Car() {

System.***out***.println("Car created!");

}

@Override

**public** **void** start() {

System.***out***.println("Car started!");

}

@Override

**public** **void** stop() {

System.***out***.println("Car stopped!");

}

@Override

**public** **void** drive() {

System.***out***.println("Driving the car with super speed!");

}

@Override

**public** String toString() {

**return** " Car ";

}

}

Boat:

**package** com.vehicle\_sys.vehiclesImple;

**import** com.vehicle\_sys.vehicles.WaterVehicle;

**public** **class** Boat **implements** WaterVehicle {

**public** Boat() {

System.***out***.println("Boat created!");

}

@Override

**public** **void** start() {

System.***out***.println("Boat started!");

}

@Override

**public** **void** stop() {

System.***out***.println("Stopped the boat!");

}

@Override

**public** **void** sail() {

System.***out***.println("Sailing the boat!!");

}

@Override

**public** String toString() {

**return** " Boat ";

}

}

VehicleOps:

**package** com.vehicle\_sys.operations;

//has methods for vehicles

**public** **interface** VehicleOps {

**void** buyCar();

**void** buyBoat();

**void** startVehicle();

**void** stopVehicle();

**void** navigate();

**void** displayAllVehicle();

}

VehicleOpsImple:

package com.vehicle\_sys.operations;

import java.util.ArrayList;

import java.util.List;

import com.vehicle\_sys.vehicles.Vehicle;

import com.vehicle\_sys.vehiclesImple.Boat;

import com.vehicle\_sys.vehiclesImple.Car;

public class VehicleOpsImple implements VehicleOps{

List<Vehicle> vehicles;

public VehicleOpsImple() {

vehicles = new ArrayList<Vehicle>();

}

@Override

public void buyCar() {

vehicles.add(new Car());

}

@Override

public void buyBoat() {

vehicles.add(new Boat());

}

@Override

public void displayAllVehicle() {

if (vehicles.isEmpty()) {

System.out.println("No vehicles bought yet!");

return;

}

System.out.println(vehicles);

}

@Override

public void startVehicle() {

if (vehicles.isEmpty()) {

System.out.println("No vehicles bought yet!");

return;

}

for(Vehicle v : vehicles) {

if (v != null) {

v.start(); //common method

}

}

}

@Override

public void stopVehicle() {

if (vehicles.isEmpty()) {

System.out.println("No vehicles bought yet!");

return;

}

for(Vehicle v : vehicles) {

if (v != null) {

v.stop(); //common methid

}

}

}

@Override

public void navigate() {

if (vehicles.isEmpty()) {

System.out.println("No vehicles bought yet!");

return;

}

for(Vehicle v : vehicles) {

if (v != null) {

//downcast

if (v instanceof Car) {

((Car) v).drive();

}

if (v instanceof Boat) {

((Boat) v).sail();;

}

}

}

}

}

TestVehicle:

package com.vehicle\_sys.tester;

import java.util.Scanner;

import com.vehicle\_sys.operations.VehicleOps;

import com.vehicle\_sys.operations.VehicleOpsImple;

public class TestVehicles {

public static void main(String[] args) {

try(Scanner sc = new Scanner(System.in)) {

System.out.println("Interface Hierrarchy Demo");

VehicleOps ops = new VehicleOpsImple();

boolean exit = false;

while (!exit) {

System.out.println("1.Buy a car\n"

+ "2.Buy a boat\n"

+ "3.Display all vehciles\n"

+ "4.Start the vehicle \n"

+ "5.Stop the vehicle\n"

+ "6.Navigate vehicle\n"

+ "0.Exit");

System.out.println("Enter choice");

switch (sc.nextInt()) {

case 1:

ops.buyCar();

break;

case 2:

ops.buyBoat();

break;

case 3:

ops.displayAllVehicle();

break;

case 4:

ops.startVehicle();

break;

case 5:

ops.stopVehicle();;

break;

case 6:

ops.navigate();

break;

case 0:

exit = true;

break;

}

}

}

}

}

**Output**

