**ASSIGNMENT 4**

**Name:- SANJOG PATEL**

**EmpId:-2607661**

**Batch: 240**

**Name**

**Exception Handling in Java**

**Solve following sub problems,**

**1. Create a class BankAccount having the members as given below:**

**Field Names:**

**• accNo (int)**

**• custName (String)**

**• accType (String)**

**• balance (float)**

**Method Description:**

**• public void deposit(float amt) : This method allows you to credit an amount into the current**

**balance. If amount is negative, throw an exception NegativeAmount toblock the operation from**

**being performed.**

**• public float getBalance() : This method returnsthe current balance. If the current balance is**

**below the minimum required balance, then throw an exception LowBalanceException**

**accordingly.**

**Have a constructorto which you will pass, accNo, custName, acctype and initial balance. And check**

**whether the balance is less than 1000 or not in case of savings account and less than 5000 in case of a**

**current account.**

**If so, then raise a LowBalanceException. In either case if the balance is negative then raise the**

**NegativeAmount exception accordingly.**

**Input 1:**

**123 John Current 4000**

**Output 1:**

**LowBalance**

**Input 2:**

**123 John Saving -900**

**Output 2:**

**NegativeAmount**

package Assignment;

public class BankAccount {

private int accNo;

private String cusName;

private String accType;

private float balance;

public BankAccount(int accNo,String cusName,String accType,float balance) throws LowBalanceException, NegativeBalanceException {

this.accNo = accNo;

this.cusName = cusName;

this.accType = accType;

this.balance = balance;

if(accType.equalsIgnoreCase("Saving")&& balance<1000) {

throw new LowBalanceException("Initial balance for saving account must be at least 1000");

}else if(accType.equalsIgnoreCase("Current")&& balance<5000) {

throw new LowBalanceException("Initial balance for current account must be at least 5000");

}else if(this.balance<0) {

throw new NegativeBalanceException();

}

}

public void deposite(float amount) throws NegativeBalanceException {

if(amount < 0) {

throw new NegativeBalanceException("deposite amount can't be -ve");

}else {

balance = balance + amount;

}

}

public float getBalance() throws LowBalanceException {

if(accType.equalsIgnoreCase("Saving")&& balance<1000) {

throw new LowBalanceException("Initial balance for saving account must be at least 1000");

}else if(accType.equalsIgnoreCase("Current")&& balance<5000) {

throw new LowBalanceException("Initial balance for current account must be at least 5000");

}

return balance;

}

public static void main(String[] args) throws LowBalanceException, NegativeBalanceException {

try {

BankAccount a1 = new BankAccount(123,"John","current",5000);

System.*out*.println("Initially Amount is "+ a1.getBalance());

a1.deposite(1000);

System.*out*.println("After Deposite Amount is "+a1.getBalance());

}catch(LowBalanceException e) {

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

}

try {

BankAccount a2 = new BankAccount(123,"John","saving",1000);

System.*out*.println("Initially Amount is "+ a2.getBalance());

a2.deposite(5000);

System.*out*.println("After Deposite Amount is "+a2.getBalance());

}catch(LowBalanceException e) {

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

}

}

}

class NegativeBalanceException extends Exception{

public NegativeBalanceException() {

super("Deposite amount can't be -ve");

}

public NegativeBalanceException(String message) {

super(message);

}

}

class LowBalanceException extends Exception{

public LowBalanceException() {

super("Balance is low !!");

}

public LowBalanceException(String m) {

super(m);

}

}

**2. WestCity Union is a cricket club that maintains an average rating of the players and provides them**

**with coins based on the rating obtained by the critics. The club can only have three critics. Create a**

**class called CricketRating with members as given below:**

**Field Names:**

**• playerName (String)**

**• critic1 (float)**

**• critic2 (float)**

**• critic3 (float**

**• avgRating (float)**

**• Coins (String)**

**Method Description:**

**• void calculateAverageRating(critic1,critic2) : This method Calculates Rating based on two**

**critics.**

**• void calculateAverageRating(critic1,critic2,critic3) : This method Calculates Rating based on**

**three critics.**

**• String calculateCoins() : This method returns the type of coin achieved by the player basedon**

**the rating.**

**• void display() : This method displays all the information.**

**The type of coin achieved by the player based on the rating is given below:**

**➢ If the avgRating is greater than or equal to 7 then the player gains gold coin.**

**➢ If the avgRating is greater than or equal to 5 and less than 7 then the player gains silver coin.**

**➢ If the avgRating is greater than or equal to 2 and less than 5 then the player gains copper**

**Coin.**

**➢ If the avgRating is less than 2 then throw a NotEligibleException.**

**Provide appropriate constructor(s) that accept values to be passed to the attributes.**

**Implementthe Tester class.**

**Input 1:**

**John 9.3 9.67 8.7**

**Output 1:**

**John 9.22 Gold**

**Input 2:**

**Henry 1.5**

**Output 2:**

**NotEligible**

package Assignment;

class NotEligibleException extends Exception{

public NotEligibleException(String m) {

super(m);

}

}

public class CricketRating {

private String playerName;

private float critic1;

private float critic2;

private float critic3;

private float avgRating;

private String coins;

public CricketRating(String playerName,float critic1,float critic2,float critic3) throws NotEligibleException {

this.playerName = playerName;

this.critic1 = critic1;

this.critic2 = critic2;

this.critic3 = critic3;

calculateAverageRating();

if(avgRating < 2) {

throw new NotEligibleException("Player is Not Eliglible for any Coin");

}

}

public float calculateAverageRating() {

avgRating = (critic1 + critic2 + critic3)/3;

return avgRating;

}

public String calculateCoins() {

if(avgRating >=7) {

coins = "Gold";

}else if(avgRating >=5) {

coins = "Silver";

}else {

coins = "Cupper";

}

return coins;

}

public static void main(String[] args) {

try {

CricketRating cr = new CricketRating("John",9.3f,9.67f,8.7f);

cr.calculateAverageRating();

String coins = cr.calculateCoins();

System.*out*.println(coins);

System.*out*.println(cr.playerName + " " + cr.avgRating + " "+ cr.coins);

}catch(NotEligibleException e) {

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

}

try {

CricketRating cr2 = new CricketRating("Henry",1.5f,0f,0f);

cr2.calculateAverageRating();

String coins = cr2.calculateCoins();

System.*out*.println(coins);

System.*out*.println(cr2.playerName + " " + cr2.avgRating + " "+ cr2.coins);

}catch(NotEligibleException e) {

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

}

}

}

**3. Cathey Bank wants to conduct examinations for the post of Probationary Officers, Assistants, and**

**Special Cadre Officers. It has rolled out an online application which is available on the Bank’s website.**

**The applicants can fill in the application form and submit it with accurate details.**

**Assuming that each Applicant is represented by the following class members:**

**Field Names:**

**• applicantName (String)**

**• postApplied (String)**

**• applicantAge (int)**

**Define a user-defined Exception CatheyBankException.**

**Design a Validator class which has following methods for validating applicant details.**

**Method Description:**

**• validate(Applicant applicant): This method receives the Applicant and calls the respective**

**methods to validate the values. If validation fails, it throws user-defined**

**exceptions CatheyBankException with the exception message as given below:**

**➢ If the violation is for applicant name, then throw InvalidNameException with the**

**message “Invalid Applicant Name”.**

**➢ If the violation is for post, then throw InvalidPostException with the message “Invalid**

**Post”.**

**➢ If the violation is for age, then throw InvalidAgeException with the message “Invalid**

**Age”.**

**If all the values are valid print the message “All details are valid”.**

**• isValidApplicantName(String name): This method validates applicantName. It cannot be null**

**or empty. If the rule is violated, then it should return false else it should return true.**

**• isValidPost(String post): This method validates the post the applicant applied for. It should**

**be one among one of the following posts: “Probationary Officer”, “Assistant”, or “Special**

**Cadre Officer”. If the rule is violated, then it should return false else it should return true.**

**• isValidAge(Integer age): This method validates the age of the applicant. It should be greater**

**than 18 years and less than 30 years. If the rule is violated, then it should return false else it**

**should return true.**

**Practice Exercise**

**August 14, 2024 | 5**

**Create a Tester class and implement it.**

**Input 1:**

**Mary Assistant 34**

**Output 1:**

**Invalid Age**

**Input 2:**

**Mary Clerk 27**

**Output 2:**

**Invalid Post**

**Input 3:**

**Probationary Officer 30**

**Output 3:**

**Invalid Applicant Name**

**Input 4:**

**Joseph Probationary Officer 30**

**Output 4:**

**All details are valid**

package Assignment;

class CatheyBankException extends Exception{

public CatheyBankException(String m) {

super(m);

}

}

class InvalidNameException extends CatheyBankException {

public InvalidNameException(String m) {

super(m);

}

}

class InvalidPostException extends CatheyBankException {

public InvalidPostException(String m) {

super(m);

}

}

class InvalidAgeException extends CatheyBankException {

public InvalidAgeException(String m) {

super(m);

}

}

class Applicant{

private String applicantName;

private String postApplied;

private int applicantAge;

public Applicant(String applicantName,String postApplied,int applicantAge) {

this.applicantName = applicantName;

this.applicantAge = applicantAge;

this.postApplied = postApplied;

}

public String getApplicantName() {

return applicantName;

}

public void setApplicantName(String applicantName) {

this.applicantName = applicantName;

}

public String getpostApplied() {

return postApplied;

}

public void setpostApplied(String postApplied) {

this.postApplied = postApplied;

}

public int getapplicantAge() {

return applicantAge;

}

public void setapplicantAge(int applicantAge) {

this.applicantAge = applicantAge;

}

}

class Validator{

public static boolean isValidApplicantName(String name) {

return name!=null && !name.isEmpty();

}

public static boolean isValidPost(String post) {

return post.equalsIgnoreCase("Probationary Officer") || post.equalsIgnoreCase("Assistant")

|| post.equalsIgnoreCase("Special Cadre Officer");

}

public static boolean isValidAge(int age) {

return age>18 && age<30;

}

public static void validate(Applicant applicant) throws CatheyBankException {

if(!*isValidApplicantName*(applicant.getApplicantName())){

throw new InvalidNameException("Invalid name");

}if(!*isValidPost*(applicant.getpostApplied())) {

throw new InvalidPostException("Invalid post");

}

if(!*isValidAge*(applicant.getapplicantAge())) {

throw new InvalidPostException("Invalid age");

}

System.*out*.println("All details are valid");

}

}

public class Tester {

public static void main(String[] args) {

Applicant applicant1 = new Applicant("Mary", "Assistant", 34);

Applicant applicant2 = new Applicant("Mary", "Clerk", 27);

Applicant applicant3 = new Applicant("","Probationary Officer", 30);

Applicant applicant4 = new Applicant("Joseph", "Probationary Officer", 30);

try {

Validator.*validate*(applicant1);

}catch(CatheyBankException e) {

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

}

try {

Validator.*validate*(applicant2);

}catch(CatheyBankException e) {

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

}

try {

Validator.*validate*(applicant3);

}catch(CatheyBankException e) {

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

}

try {

Validator.*validate*(applicant4);

}catch(CatheyBankException e) {

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

}

}

}