*Name: Krithika Swaminathan*
*Roll No.: 205001057*

# Assignment 5 – Exception Handling

**Q1: Create a class named "Person" which consists of name, age, aadharnumber. Create methods getInput(), display(), canVote(), hasAadhar(). Create and handle the following Exceptions.**
**a. For age -> if you give alphabets then throw NumberFormatException (Check for the condition explicitly and throw builtin exception)**
**b. For voting -> if age is less than 18 then throw MinorCitizenException (Check for the condition explicitly and throw user-defined exception)**
**c. For aadhar -> if no valid aadhar then throw NullPointerException (Check for the condition explicitly and throw builtin exception)**

## Code:

```
import java.util.Scanner;

class MinorCitizenException extends Exception {
	String message;
	MinorCitizenException (String m) {
		message = m;
		}
	}

class Person {
	//data members
	String name;
	int age;
	String aadharnumber;
	//member functions
	void getInput (String n, int a, String an) {
		name = n;
		age = a;
		aadharnumber = an;
		}
	void display() {
		System.out.println("Name: "+name);
		System.out.println("Age: "+age);
		System.out.println("Aadhar number: "+aadharnumber);
		}
	boolean canVote() {
		if (age>=18)
			return true;
		return false;
		}
	boolean hasAadhar() {
		if (aadharnumber.length() == 12)
			return true;
		return false;
```

```
		}
	}

class TestPerson {
	public static void main (String arg[]) {
		Scanner sc = new Scanner(System.in);

		System.out.println("__PERSON__");
		Person p = new Person();

		String name;
		int age = -1;
		String aadharno;
		//to test exception handling

		System.out.print("Enter name: ");
		name = sc.nextLine();
		//to validate user-input age
		try {
			System.out.print("Enter age: ");
			String input = sc.next();
			try {
				age = Integer.valueOf(input);
				}
			catch (NumberFormatException e) {
				System.out.println("Error! Age should be a number");
				System.exit(0);
				}
			if (age<0)
				throw new NumberFormatException("Age should be a number");
			}
		//to catch number input exception
		catch (NumberFormatException e) {
			System.out.println("Error! Age must be a positive integer.");
			System.exit(0);
			}
		System.out.print("Enter aadhar number: ");
		aadharno = sc.next();
		//if datatype of all details correct, build record
		p.getInput(name,age,aadharno);

		//to catch minor citizen exception
		try {
			if (!p.canVote())
				throw new MinorCitizenException("Error! Citizen must be above 18 to
vote.");
			}
		catch (MinorCitizenException e) {
			System.out.println(e.message);
```

```
		}

	//to validate aadhar number using built-in exception
	try {
		if (!p.hasAadhar())
			throw new NullPointerException("Aadhar no. must be 12 digits. ");
		}
	catch (NullPointerException e) {
		System.out.println("Error! Valid aadhar number should be entered.");
		System.exit(0);
		}

	//if all details correct, enter into record
	System.out.println("Record of person entered into system.");
	}
}
```

## Output:

```
kri@kri-ubuntu:~/workspace$ javac TestPerson.java
kri@kri-ubuntu:~/workspace$ java TestPerson
__PERSON__
Enter name: Meera
Enter age: 20
Enter aadhar number: 938293849282
Record of person entered into system.
kri@kri-ubuntu:~/workspace$ java TestPerson
__PERSON__
Enter name: Ravi
Enter age: 12
Enter aadhar number: 798294928292
Error! Citizen must be above 18 to vote.
Record of person entered into system.
kri@kri-ubuntu:~/workspace$ java TestPerson
__PERSON__
Enter name: Sona
Enter age: 24
Enter aadhar number: 34029192
Error! Valid aadhar number should be entered.
kri@kri-ubuntu:~/workspace$ java TestPerson
__PERSON__
Enter name: Valli
Enter age: t
Error! Age should be a number
kri@kri-ubuntu:~/workspace$ java TestPerson
__PERSON__
Enter name: Vishnu
Enter age: -4
Error! Age must be a positive integer.
```

*Name: Krithika Swaminathan*
*Roll No.: 205001057*

**Q2: Create a class named "Account" which contains name, acct_num, branch, balance, PAN_num. Create functions for deposit and withdrawal. Write user-defined exceptions for the following conditions:**
**a. In deposit function, if the customer deposits money more than 25000, then throw the user defined exception "PANRequiredException" and get the PAN number and proceed the deposit.**
**b. In withdrawal function, if the customer requesting some money, check on withdrawal will it satisfy the minimum_bal amount and throw the "MinBalRequiredException" exception. If the withdrawal amount is more than the balance amount then throw "NotEnougMoneyInAccountException".**
**c. Search for a particular acct_num. If not present then throw "AccountNotFoundException".**
**d. On PAN number entry check the format of 10 characters. First 5 characters then 4 numbers and then 1 character. If the format not matched then throw "PANFormatMismatchException".**

## Code:

```
import java.util.Scanner;

class PANRequiredException extends Exception {
	String message;
	PANRequiredException (String m) {
		message = m;
		}
	}

class MinBalRequiredException extends Exception {
	String message;
	MinBalRequiredException (String m) {
		message = m;
		}
	}

class NotEnoughMoneyInAccountException extends Exception {
	String message;
	NotEnoughMoneyInAccountException (String m) {
		message = m;
		}
	}

class AccountNotFoundException extends Exception {
	String message;
	AccountNotFoundException (String m) {
		message = m;
		}
	}
```

```
class PANFormatMismatchException extends Exception {
	String message;
	PANFormatMismatchException (String m) {
		message = m;
		}
	}

class Account {
	//data members
	String name;
	int acct_num;
	String branch;
	int balance;
	int minimum_bal = 2000;
	String PAN_num;
	//consructor
	Account (String n, int a, String b, int bal) {
		name = n;
		acct_num = a;
		branch = b;
		balance = bal;
		}
	//member functions
	void deposit (int amt) {
		balance += amt;
		}
	void withdraw (int amt) {
		balance -= amt;
		}
	}

class TestAccount {
	public static void main (String arg[]) {
		Scanner sc = new Scanner(System.in);
		String name, branch, PAN;
		int aNum, balance, amount;
		Account[] acc = new Account[2];

		//account object 1
		System.out.println("__ACCOUNT-1__");
			//acc[0] = new Account();
		System.out.print("Enter name: ");
		name = sc.nextLine();
		System.out.print("Enter account number: ");
		aNum = sc.nextInt();
		sc.nextLine();
		System.out.print("Enter branch: ");
		branch = sc.nextLine();
		System.out.print("Enter balance: ");
```

```
		balance = sc.nextInt();

		acc[0] = new Account(name,aNum,branch,balance);

		//to check for exceptions
			//deposit
		System.out.print("Enter amount to deposit: ");
		amount = sc.nextInt();
		try {
			if (amount > 25000)
				throw new PANRequiredException("PAN number required for deposits
greater than 25000.");
			acc[0].deposit(amount);
			}
		catch (PANRequiredException e) {
			System.out.println(e.message);
			System.out.print("Enter PAN number: ");
			PAN = sc.next();
			//PAN number mismatch
			if (PAN.length() != 10) {
				try {
					throw new PANFormatMismatchException("PAN number must have
10 characters.");
					}
				catch (PANFormatMismatchException e0) {
					System.out.println(e0.message);
					}
				}

			else {
				for (int i=0; i<10; i++) {
					if (i>=5 && i<=8) {
						try {
							int pan = Integer.valueOf(PAN.charAt(i));
							throw new PANFormatMismatchException("Invalid!
PAN number format is cccccnnnnc (c-char, n-num");
							}
						catch (PANFormatMismatchException e1) {
							System.out.println("Invalid! PAN number format is
cccccnnnnc (c-char, n-num)");
							}
						}
					else {
						try {
							if (!( ('A'<=PAN.charAt(i) && PAN.charAt(i)<='Z') ||
('a'<=PAN.charAt(i) && PAN.charAt(i)<='z') ))
								throw new
PANFormatMismatchException("Invalid! PAN number format is cccccnnnnc (c-char, n-num");
							}
```

```
                                        catch (PANFormatMismatchException e2) {
                                                System.out.println(e2.message);
                                                }
                                        }
                                }
                        }
                acc[0].PAN_num = PAN;
                }

                //withdrawal
        System.out.print("Enter amount to withdraw: ");
        amount = sc.nextInt();
        balance = acc[0].balance;
        try {
                if (amount > balance)
                        throw new NotEnoughMoneyInAccountException("Not enough balance in
account.");
                else if (balance-amount < acc[0].minimum_bal)
                        throw new MinBalRequiredException("Minimum balance of 2000 is required
in account.");

                }
        catch (NotEnoughMoneyInAccountException e) {
                System.out.println("Cannot withdraw! "+e.message);
                }
        catch (MinBalRequiredException e) {
                System.out.println("Cannot withdraw! "+e.message);
                }

        //account object 2
        acc[1] = new Account("Hannah",230,"Adyar",20000);

        //to search for account number
        System.out.print("Enter account number to search for: ");
        int n = sc.nextInt();
        int flag = 0;
        for (int i=0; i<acc.length; i++) {
                if (acc[i].acct_num == n) {
                        flag = 1;
                        }
                try {
                        if (flag==0)
                                throw new AccountNotFoundException("Account number not found
in records.");
                        }
                }
        catch (AccountNotFoundException e) {
                        System.out.println(e.message);
                        }
```

```
        }
    }
```

## Output:

```
kri@kri-ubuntu:~/workspace$ javac TestAccount.java
kri@kri-ubuntu:~/workspace$ java TestAccount
__ACCOUNT-1__
Enter name: Mina
Enter account number: 124
Enter branch: Adyar
Enter balance: 26000
Enter amount to deposit: 10000
Enter amount to withdraw: 3000
Enter account number to search for: 130
Account number not found in records.
```