Group Members:

Akshay Dake(MT2017011)

AmarPrakash Mishra(MT2017015)

Problem Statement:

Mutation-source-code: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function. The mutated program needs to be strongly killed by the de- signed test cases. At least three different mutation operators should be used.

What is Mutation Testing?

Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code.

Mutation Testing Types:

* **Value Mutations:** An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.
* **Decision Mutations:** The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)
* **Statement Mutations:** Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

Testing java file

package com.beyondscheme.pitest;

import org.junit.Before;

import org.junit.Test;

import java.util.ArrayList;

import java.util.List;

import static org.junit.Assert.assertEquals;

public class TicketPriceCalculatorTest {

private static int ADULT\_TICKET\_PRICE\_BUS = 40;

private static int CHILD\_TICKER\_PRICE\_BUS = 20;

private static int ADULT\_TICKET\_PRICE\_TRAIN = 20;

private static int CHILD\_TICKER\_PRICE\_TRAIN = 10;

private static int ADULT\_TICKET\_PRICE\_TAXI = 80;

private static int CHILD\_TICKER\_PRICE\_TAXI = 50;

private static int ADULT\_TICKET\_PRICE\_AIRPLANE = 4000;

private static int CHILD\_TICKER\_PRICE\_AIRPLANE = 2000;

private static int ADULT\_TICKET\_PRICE\_SHIP = 4000;

private static int CHILD\_TICKER\_PRICE\_SHIP = 2000;

private static int ADULT\_TICKET\_PRICE\_MOVIE = 4000;

private static int CHILD\_TICKER\_PRICE\_MOVIE = 2000;

private TicketPriceCalculator calculator;

@Before

public void setUp() {

calculator = new TicketPriceCalculator();

}

@Test

public void calculatePriceForOneAdultBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger passenger = new Passenger(20);

passengers.add(passenger);

int[] arr = {10,30,35,28,22,25,29,50,78,44};

int n=10;

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals(ADULT\_TICKET\_PRICE\_BUS, price, 0);

}

/\*

@Test

public void sortPassengerAgeBUS() {

int[] arr = {10,30,35,28,22,25,29,50,78,44};

int n=10;

int[] age = calculator.mergeSort(arr,n);

assertEquals(arr,age);

}

\*/

@Test

public void calculatePriceForChildBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(15);

passengers.add(childPassenger);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals(CHILD\_TICKER\_PRICE\_BUS, price, 0);

}

@Test

public void calculatePriceForFamilyBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger3 = new Passenger(12);

Passenger childPassenger4 = new Passenger(4);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger3);

passengers.add(childPassenger4);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_BUS + 2 \* CHILD\_TICKER\_PRICE\_BUS) \*

(1 - TicketPriceCalculator.FAMILY\_DISCOUNT\_BUS), price, 0);

}

@Test

public void calculatePriceForChildNarrowCaseBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(18);

passengers.add(childPassenger);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals(CHILD\_TICKER\_PRICE\_BUS, price, 0);

}

@Test

public void calculatePriceForFreeTicketNarrowCaseBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(3);

passengers.add(childPassenger);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals(0, price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithAdultsBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger1);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_BUS + CHILD\_TICKER\_PRICE\_BUS), price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithChildrenBUS() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

Passenger childPassenger2 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(childPassenger1);

passengers.add(childPassenger2);

double price = calculator.calculatePriceBus(passengers, ADULT\_TICKET\_PRICE\_BUS, CHILD\_TICKER\_PRICE\_BUS);

assertEquals((ADULT\_TICKET\_PRICE\_BUS + 2 \* CHILD\_TICKER\_PRICE\_BUS), price, 0);

}

@Test

public void calculatePriceForOneAdultTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger passenger = new Passenger(20);

passengers.add(passenger);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(ADULT\_TICKET\_PRICE\_TRAIN, price, 0);

}

@Test

public void calculatePriceForChildTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(15);

passengers.add(childPassenger);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(CHILD\_TICKER\_PRICE\_TRAIN, price, 0);

}

@Test

public void calculatePriceForFamilyTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger3 = new Passenger(12);

Passenger childPassenger4 = new Passenger(4);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger3);

passengers.add(childPassenger4);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_TRAIN + 2 \* CHILD\_TICKER\_PRICE\_TRAIN) \*

(1 - TicketPriceCalculator.FAMILY\_DISCOUNT\_TRAIN), price, 0);

}

@Test

public void calculatePriceForChildNarrowCaseTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(18);

passengers.add(childPassenger);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(CHILD\_TICKER\_PRICE\_TRAIN, price, 0);

}

@Test

public void calculatePriceForFreeTicketNarrowCaseTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(3);

passengers.add(childPassenger);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(0, price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithAdultsTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger1);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_TRAIN + CHILD\_TICKER\_PRICE\_TRAIN), price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithChildrenTRAIN() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

Passenger childPassenger2 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(childPassenger1);

passengers.add(childPassenger2);

double price = calculator.calculatePriceTrain(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals((ADULT\_TICKET\_PRICE\_TRAIN + 2 \* CHILD\_TICKER\_PRICE\_TRAIN), price, 0);

}

@Test

public void calculatePriceForOneAdultTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger passenger = new Passenger(20);

passengers.add(passenger);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TRAIN, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(ADULT\_TICKET\_PRICE\_TRAIN, price, 0);

}

@Test

public void calculatePriceForChildTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(15);

passengers.add(childPassenger);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TRAIN);

assertEquals(CHILD\_TICKER\_PRICE\_TRAIN, price, 0);

}

@Test

public void calculatePriceForFamilyTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger3 = new Passenger(12);

Passenger childPassenger4 = new Passenger(4);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger3);

passengers.add(childPassenger4);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TAXI);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_TAXI + 2 \* CHILD\_TICKER\_PRICE\_TAXI) \*

(1 - TicketPriceCalculator.FAMILY\_DISCOUNT\_TAXI), price, 0);

}

@Test

public void calculatePriceForChildNarrowCaseTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(18);

passengers.add(childPassenger);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TAXI);

assertEquals(CHILD\_TICKER\_PRICE\_TAXI, price, 0);

}

@Test

public void calculatePriceForFreeTicketNarrowCaseTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(3);

passengers.add(childPassenger);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TAXI);

assertEquals(0, price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithAdultsTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger1);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TAXI);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_TAXI + CHILD\_TICKER\_PRICE\_TAXI), price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithChildrenTAXI() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

Passenger childPassenger2 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(childPassenger1);

passengers.add(childPassenger2);

double price = calculator.calculatePriceTaxi(passengers, ADULT\_TICKET\_PRICE\_TAXI, CHILD\_TICKER\_PRICE\_TAXI);

assertEquals((ADULT\_TICKET\_PRICE\_TAXI + 2 \* CHILD\_TICKER\_PRICE\_TAXI), price, 0);

}

@Test

public void calculatePriceForOneAdultShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger passenger = new Passenger(20);

passengers.add(passenger);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals(ADULT\_TICKET\_PRICE\_SHIP, price, 0);

}

@Test

public void calculatePriceForChildShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(15);

passengers.add(childPassenger);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals(CHILD\_TICKER\_PRICE\_SHIP, price, 0);

}

@Test

public void calculatePriceForFamilyShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger3 = new Passenger(12);

Passenger childPassenger4 = new Passenger(4);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger3);

passengers.add(childPassenger4);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_SHIP + 2 \* CHILD\_TICKER\_PRICE\_SHIP) \*

(1 - TicketPriceCalculator.FAMILY\_DISCOUNT\_SHIP), price, 0);

}

@Test

public void calculatePriceForChildNarrowCaseShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(18);

passengers.add(childPassenger);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_AIRPLANE);

assertEquals(CHILD\_TICKER\_PRICE\_AIRPLANE, price, 0);

}

@Test

public void calculatePriceForFreeTicketNarrowCaseShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(3);

passengers.add(childPassenger);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals(0, price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithAdultsShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger1);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_SHIP, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_SHIP + CHILD\_TICKER\_PRICE\_SHIP), price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithChildrenShip() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

Passenger childPassenger2 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(childPassenger1);

passengers.add(childPassenger2);

double price = calculator.calculatePriceShip(passengers, ADULT\_TICKET\_PRICE\_AIRPLANE, CHILD\_TICKER\_PRICE\_SHIP);

assertEquals((ADULT\_TICKET\_PRICE\_SHIP + 2 \* CHILD\_TICKER\_PRICE\_SHIP), price, 0);

}

@Test

public void calculatePriceForOneAdultMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger passenger = new Passenger(20);

passengers.add(passenger);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals(ADULT\_TICKET\_PRICE\_MOVIE, price, 0);

}

@Test

public void calculatePriceForChildMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(15);

passengers.add(childPassenger);

double price = calculator.calculatePriceAirPlane(passengers, ADULT\_TICKET\_PRICE\_AIRPLANE, CHILD\_TICKER\_PRICE\_AIRPLANE);

assertEquals(CHILD\_TICKER\_PRICE\_AIRPLANE, price, 0);

}

@Test

public void calculatePriceForFamilyMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger3 = new Passenger(12);

Passenger childPassenger4 = new Passenger(4);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger3);

passengers.add(childPassenger4);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_MOVIE + 2 \* CHILD\_TICKER\_PRICE\_MOVIE) \*

(1 - TicketPriceCalculator.FAMILY\_DISCOUNT\_MOVIE), price, 0);

}

@Test

public void calculatePriceForChildNarrowCaseMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(18);

passengers.add(childPassenger);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals(CHILD\_TICKER\_PRICE\_MOVIE, price, 0);

}

@Test

public void calculatePriceForFreeTicketNarrowCaseMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger childPassenger = new Passenger(3);

passengers.add(childPassenger);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals(0, price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithAdultsMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger adultPassenger2 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(adultPassenger2);

passengers.add(childPassenger1);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals((2 \* ADULT\_TICKET\_PRICE\_MOVIE + CHILD\_TICKER\_PRICE\_MOVIE), price, 0);

}

@Test

public void shouldNotCalculatePriceForFamilyEdgeCaseWithChildrenMovie() {

List<Passenger> passengers = new ArrayList<>();

Passenger adultPassenger1 = new Passenger(20);

Passenger childPassenger1 = new Passenger(12);

Passenger childPassenger2 = new Passenger(12);

passengers.add(adultPassenger1);

passengers.add(childPassenger1);

passengers.add(childPassenger2);

double price = calculator.calculatePriceMovie(passengers, ADULT\_TICKET\_PRICE\_MOVIE, CHILD\_TICKER\_PRICE\_MOVIE);

assertEquals((ADULT\_TICKET\_PRICE\_MOVIE + 2 \* CHILD\_TICKER\_PRICE\_MOVIE), price, 0);

}

}

JAVA FILE##################################################################

package com.akshaydake.pitest;

import java.util.List;

public class TicketPriceCalculator {

private static int ADULT\_AGE\_BUS = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_BUS = 3;

private static int SENIOR\_CITIZEN\_BUS = 60;

private static int ARMY\_OFFICER\_BUS = 18;

public static double FAMILY\_DISCOUNT\_BUS= 0.05;

private static int ADULT\_AGE\_TRAIN = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_TRAIN = 3;

private static int SENIOR\_CITIZEN\_TRAIN = 60;

private static int ARMY\_OFFICER\_TRAIN = 18;

public static double FAMILY\_DISCOUNT\_TRAIN= 0.25;

private static int ADULT\_AGE\_AIRPLANE = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_AIRPLANE = 3;

private static int SENIOR\_CITIZEN\_AIRPLANE = 60;

private static int ARMY\_OFFICER\_AIRPLANE = 18;

public static double FAMILY\_DISCOUNT\_AIRPLANE= 1;

private static int ADULT\_AGE\_TAXI = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_TAXI = 3;

private static int SENIOR\_CITIZEN\_TAXI = 60;

private static int ARMY\_OFFICER\_TAXI = 18;

public static double FAMILY\_DISCOUNT\_TAXI= 0.15;

private static int ADULT\_AGE\_SHIP = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_SHIP = 3;

private static int SENIOR\_CITIZEN\_SHIP = 60;

private static int ARMY\_OFFICER\_SHIP = 18;

public static double FAMILY\_DISCOUNT\_SHIP= 0.05;

private static int ADULT\_AGE\_MOVIE = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_MOVIE = 3;

private static int SENIOR\_CITIZEN\_MOVIE = 60;

private static int ARMY\_OFFICER\_MOVIE = 18;

public static double FAMILY\_DISCOUNT\_MOVIE= 0.05;

private static int ADULT\_AGE\_HOTEL = 18;

private static int FREE\_TICKET\_AGE\_BELOW\_HOTEL = 3;

private static int SENIOR\_CITIZEN\_HOTEL = 60;

private static int ARMY\_OFFICER\_HOTEL = 18;

public static double FAMILY\_DISCOUNT\_HOTEL= 0.05;

public double calculatePriceBus(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice ) {

int totalPriceBus = 0;

int childrenCounterBus = 0;

int adultCounterBus = 0;

double resultBus;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_BUS) {

totalPriceBus += adultTicketPrice;

adultCounterBus++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_BUS) {

totalPriceBus += childTicketPrice;

childrenCounterBus++;

}

}

if (childrenCounterBus > 1 && adultCounterBus > 1) {

resultBus = (1 - FAMILY\_DISCOUNT\_BUS) \* totalPriceBus;

}

else {

resultBus = totalPriceBus;

}

//sortBusPassengerAge(arr,n);

return resultBus;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortBus(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeBus(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeBus(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePriceTrain(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceTrain = 0;

int childrenCounterTrain = 0;

int adultCounterTrain = 0;

double resultTrain;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_TRAIN) {

totalPriceTrain += adultTicketPrice;

adultCounterTrain++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_TRAIN) {

totalPriceTrain += childTicketPrice;

childrenCounterTrain++;

}

}

if (childrenCounterTrain > 1 && adultCounterTrain > 1) {

resultTrain = (1 - FAMILY\_DISCOUNT\_TRAIN) \* totalPriceTrain;

}

else {

resultTrain = totalPriceTrain;

}

return resultTrain;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortTrain(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeTrain(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeTrain(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePriceAirPlane(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPrice = 0;

int childrenCounter = 0;

int adultCounter = 0;

double result;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_AIRPLANE) {

totalPrice += adultTicketPrice;

adultCounter++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_AIRPLANE) {

totalPrice += childTicketPrice;

childrenCounter++;

}

}

if (childrenCounter > 1 && adultCounter > 1) {

result = (1 - FAMILY\_DISCOUNT\_AIRPLANE) \* totalPrice;

}

else {

result = totalPrice;

}

return result;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortAirPlane(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeAirPlane(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeAirPlane(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePriceTaxi(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceTaxi = 0;

int childrenCounterTaxi = 0;

int adultCounterTaxi = 0;

double resultTaxi;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_TAXI) {

totalPriceTaxi += adultTicketPrice;

adultCounterTaxi++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_TAXI) {

totalPriceTaxi += childTicketPrice;

childrenCounterTaxi++;

}

}

if (childrenCounterTaxi > 1 && adultCounterTaxi > 1) {

resultTaxi = (1 - FAMILY\_DISCOUNT\_TAXI) \* totalPriceTaxi;

}

else {

resultTaxi = totalPriceTaxi;

}

return resultTaxi;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortTaxi(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeTaxi(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeTaxi(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePriceMovie(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceMovie = 0;

int childrenCounterMovie = 0;

int adultCounterMovie = 0;

double resultMovie;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_MOVIE) {

totalPriceMovie += adultTicketPrice;

adultCounterMovie++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_MOVIE) {

totalPriceMovie+= childTicketPrice;

childrenCounterMovie++;

}

}

if (childrenCounterMovie > 1 && adultCounterMovie > 1) {

resultMovie = (1 - FAMILY\_DISCOUNT\_MOVIE) \* totalPriceMovie;

}

else {

resultMovie = totalPriceMovie;

}

return resultMovie;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortMovie(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeMovie(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeMovie(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePriceHotel(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceMovie = 0;

int childrenCounterMovie = 0;

int adultCounterMovie = 0;

double resultMovie;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_MOVIE) {

totalPriceMovie += adultTicketPrice;

adultCounterMovie++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_MOVIE) {

totalPriceMovie+= childTicketPrice;

childrenCounterMovie++;

}

}

if (childrenCounterMovie > 1 && adultCounterMovie > 1) {

resultMovie = (1 - FAMILY\_DISCOUNT\_MOVIE) \* totalPriceMovie;

}

else {

resultMovie = totalPriceMovie;

}

return resultMovie;

}

public double calculatePriceShip(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceMovie = 0;

int childrenCounterMovie = 0;

int adultCounterMovie = 0;

double resultMovie;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_MOVIE) {

totalPriceMovie += adultTicketPrice;

adultCounterMovie++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_MOVIE) {

totalPriceMovie+= childTicketPrice;

childrenCounterMovie++;

}

}

if (childrenCounterMovie > 1 && adultCounterMovie > 1) {

resultMovie = (1 - FAMILY\_DISCOUNT\_MOVIE) \* totalPriceMovie;

}

else {

resultMovie = totalPriceMovie;

}

return resultMovie;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortShip(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergeBus(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergeShip(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

public double calculatePricePark(List<Passenger> passengers, int adultTicketPrice, int childTicketPrice) {

int totalPriceMovie = 0;

int childrenCounterMovie = 0;

int adultCounterMovie = 0;

double resultMovie;

int seniorCitizen=0;

for (Passenger passenger : passengers) {

if(passenger.getAge() >150)

{

System.out.println("Not Valid");

}

if(passenger.getAge() < 0)

{

System.out.println("Not Valid");

}

if(passenger.getAge() >60 && passenger.getAge()<100)

{

System.out.println("Person is senior citizen");

seniorCitizen++;

}

if (passenger.getAge() > ADULT\_AGE\_MOVIE) {

totalPriceMovie += adultTicketPrice;

adultCounterMovie++;

}

else if (passenger.getAge() > FREE\_TICKET\_AGE\_BELOW\_MOVIE) {

totalPriceMovie+= childTicketPrice;

childrenCounterMovie++;

}

}

if (childrenCounterMovie > 1 && adultCounterMovie > 1) {

resultMovie = (1 - FAMILY\_DISCOUNT\_MOVIE) \* totalPriceMovie;

}

else {

resultMovie = totalPriceMovie;

}

return resultMovie;

}

/\* Iterative mergesort function to sor

t arr[0...n-1] \*/

int[] mergeSortPark(int arr[], int n)

{

// For current size of subarrays to

// be merged curr\_size varies from

// 1 to n/2

int curr\_size;

// For picking starting index of

// left subarray to be merged

int left\_start;

// Merge subarrays in bottom up

// manner. First merge subarrays

// of size 1 to create sorted

// subarrays of size 2, then merge

// subarrays of size 2 to create

// sorted subarrays of size 4, and

// so on.

for (curr\_size = 1; curr\_size <= n-1;

curr\_size = 2\*curr\_size)

{

// Pick starting point of different

// subarrays of current size

for (left\_start = 0; left\_start < n-1;

left\_start += 2\*curr\_size)

{

// Find ending point of left

// subarray. mid+1 is starting

// point of right

int mid = left\_start + curr\_size - 1;

int right\_end = Math.min(left\_start

+ 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid]

// & arr[mid+1...right\_end]

mergePark(arr, left\_start, mid, right\_end);

}

}

return arr;

}

void mergePark(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

}