public abstract class Vehicle {

private double maxSpeed;

public Vehicle(double maxSpeed) {

this.maxSpeed = maxSpeed;

}

public double getMaxSpeed() {

return maxSpeed;

}

public abstract boolean startTrip(Terrain t);

public abstract boolean endTrip(Terrain t);

public abstract boolean move(Terrain t);

}

public class Car extends Vehicle {

private double turboBoost;

public Car(double maxSpeed, double turboBoost) {

// call superclass (Vehicle) constructor

super(maxSpeed);

this.turboBoost = turboBoost

}

public boolean endTrip(Terrain t) {

if ( t == Terrain.AIRPORT || t == Terrain.MARINA ) {

return true;

} else {

return false;

}

}

public boolean move(Terrain t) {

if ( t == Terrain.AIRPORT || t == Terrain.MARINA

|| t == Terrain.ROAD ) {

return true;

} else {

return false;

}

}

public boolean startTrip(Terrain t) {

if ( t == Terrain.AIRPORT || t == Terrain.MARINA ) {

return true;

} else {

return false;

}

}

public double getTurboSpeed() {

return getMaxSpeed()\*turboBoost;

}

}

public class Trip {

private Terrain[] hops;

public Trip(int numHops) {

if (numHops < 2) {

throw new IllegalArgumentException("Trips must have at least a

start and finish");

}

this.hops = new Terrain[numHops];

}

public void setHop(int hop, Terrain t) {

hops[hop] = t;

}

public boolean isTripPossible(Vehicle v) {

// Check the first hop

if (!v.startTrip(hops[0])) {

return false;

}

// Check all hops between the first and last

for (int i = 1; i < hops.length - 1; i++) {

if (!v.move(hops[i])) {

return false;

}

}

// Check the last hop

if (!v.endTrip(hops[hops.length - 1])) {

return false;

}

// success!

return true;

}

}