# Answers:

# Part A: Abstract Classes

abstract class Vehicle {   
 String brand, model;  
  
 Vehicle(String b, String m) {   
 brand = b;   
 model = m;   
 }  
  
 abstract void move(); // must be implemented in subclass  
  
 void fuelUp() { System.out.println("Fueling up..."); }  
}  
  
class Car extends Vehicle {  
 Car(String b, String m) { super(b, m); }  
 void move() { System.out.println(brand + " " + model + " drives."); } // implementation  
}  
  
public class TestA {  
 public static void main(String[] args) {  
 // Vehicle v = new Vehicle("Generic","V1"); // ERROR: cannot instantiate abstract class  
 Car c = new Car("Toyota","Camry");  
 c.move();  
 c.fuelUp();  
 }  
}

# Part B: Abstract Methods

abstract class Vehicle { ...   
 abstract void move();  
 abstract void honk(); // added abstract method  
}  
  
class Car extends Vehicle {  
 Car(String b, String m) { super(b, m); }  
 void move() { System.out.println("Driving..."); }  
 void honk() { System.out.println("Beep!"); } // must implement honk()  
}  
  
// If honk() is not implemented → ERROR: Car is not abstract and does not override abstract method honk()

# Part C: Interfaces

interface Drivable {   
 void accelerate();  
 void brake();  
}  
  
class Car extends Vehicle implements Drivable {  
 Car(String b, String m) { super(b, m); }  
 void move() { System.out.println("Driving..."); }  
 public void accelerate() { System.out.println("Car speeds up"); }  
 public void brake() { System.out.println("Car slows down"); }  
}  
  
interface Electric { void charge(int minutes); }  
  
class ElectricCar extends Vehicle implements Drivable, Electric {  
 ElectricCar(String b, String m) { super(b, m); }  
 void move() { System.out.println("Silent drive"); }  
 public void accelerate() { System.out.println("Zoom!"); }  
 public void brake() { System.out.println("Stops"); }  
 public void charge(int minutes) { System.out.println("Charging for " + minutes + " mins"); }  
}

# Part D: Why Interfaces?

static void driveForASecond(Drivable d) {  
 d.accelerate();  
 d.brake();  
}  
  
// Works with Car and ElectricCar because both implement Drivable

# Part E: Mixed Practice

class Truck extends Vehicle implements Drivable {  
 Truck(String b, String m) { super(b, m); }  
 void move() { System.out.println("Truck hauling..."); }  
 void honk() { System.out.println("Honk honk!"); }  
 public void accelerate() { System.out.println("Truck speeds up"); }  
 public void brake() { System.out.println("Truck slows down"); }  
}  
  
public class TestE {  
 public static void main(String[] args) {  
 Truck t = new Truck("Ford","F-150");  
 Vehicle v = t; // upcasting to Vehicle  
 Drivable d = t; // upcasting to Drivable  
 v.move();  
 v.honk();  
 d.accelerate();  
 d.brake();  
 }  
}

# Exit Ticket Answers

1) Abstract classes may have unimplemented methods; you can’t make an incomplete object.  
2) Subclasses must implement all abstract methods or declare themselves abstract.  
3) Interfaces allow multiple unrelated classes to share the same behavior contract.  
4) Abstract class = shared state + some code; Interface = just shared behavior (multiple allowed).