CPS 181 Chapter 3 Worksheet: Abstract Classes & Interfaces

# Part A: Abstract Classes

1) Create an abstract class Vehicle with fields brand, model.

2) Add an abstract method move().

3) Add a concrete method fuelUp().

4) Create a subclass Car that extends Vehicle and implements move().

5) In main, try: Vehicle v = new Vehicle(); What happens?

6) Create a Car and call its methods.

# Part B: Abstract Methods

1) Add another abstract method honk() to Vehicle.

2) Update subclasses to implement honk().

3) Try compiling without one implementation. What happens?

# Part C: Interfaces

1) Create an interface Drivable with accelerate() and brake().

2) Have Car implement Drivable.

3) In main, use a Drivable reference for a Car.

4) Create an interface Electric with charge().

5) Make an ElectricCar class that implements both.

# Part D: Why Interfaces?

1) Write a method driveForASecond(Drivable d) that calls accelerate() and brake().

2) Test it with both a Car and an ElectricCar.

# Part E: Mixed Practice

1) Keep Vehicle abstract with move() and honk().

2) Keep Drivable as an interface.

3) Create a Truck that extends Vehicle and implements Drivable.

4) Test all methods in main.

# Exit Ticket

1) Why can’t you instantiate an abstract class?

2) What must a subclass do with abstract methods?

3) What’s the main benefit of using an interface?

4) When would you use an abstract class vs an interface?

# 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).