Advanced Object Oriented Programming – Inheritance,

Polymorphism, Override vs. Overload, instanceOf

1. Study the following classes (all of which compile correctly) and then answer the questions that follow them.

class Vehicle {

private int numWheels;

public Vehicle (int nw) {

numWheels = nw;

}

public int getNumWheels() {

return numWheels;

}

public boolean equals(Vehicle other) {

return other != null &&

numWheels == other.numWheels;

}

public boolean isBig() {

return numWheels > 4;

}

}

class Truck extends Vehicle {

private int loadCapacity;

public Truck (int nw, int lc) {

super(nw);

loadCapacity = lc;

}

public int getLoadCapacity() {

return loadCapacity;

}

public boolean equals(Truck other) {

return other != null &&

loadCapacity == other.loadCapacity;

}

}

class Car extends Vehicle {

private int numSeats;

public Car (int ns) {

super(4);

numSeats = ns;

}

public int getNumSeats() {

return numSeats;

}

public boolean isBig() {

return numSeats > 5;

}

}

Consider each of the following fragments to be in a main method that uses these classes. All the constructors are valid. The other statements fall into one of the following categories.

1. The statement will compile and execute correctly.
2. The statement will not compile but it can be repaired by a cast.
3. The statement will not compile and cannot be repaired by a cast.
4. The statement will compile but fails to execute correctly.

For each fragment, identify the category to which the second statement belongs.

1. Vehicle v1 = new Vehicle(4);

int nw = v1.getNumWheels();

**i)**

1. Car c1 = new Car(6);

System.out.println(c1.getNumWheels());

**i)**

1. Vehicle v2 = new Car(4);

int capacity = v2.getNumSeats();

**ii)**

1. Truck t1 = new Truck(18, 14000);

int loadLimit = ((Vehicle)t1).getNumSeats();

**iii)**

1. Car c2 = new Car(5);

int maxLoad = (((Truck)c2).getLoadCapacity());

**iii)**

1. Vehicle v3 = new Vehicle(4);

System.out.println(((Car)v3).getNumSeats());

**iv)**

1. Vehicle v4 = new Truck(4, 1200);

System.out.println(((Truck)v4).getNumWheels());

**i)**

1. Truck t2 = new Truck(10, 8000);

Car c3 = (Car)t2;

**iii) Not in the same hierarchy**

Assume the following fragment has been run,

Vehicle v1 = new Vehicle(4);

Vehicle v2 = new Vehicle(4);

Vehicle v3 = new Vehicle(6);

Vehicle v4 = new Truck(10, 500);

Truck t1 = new Truck (10, 1000);

Truck t2 = new Truck (10, 700);

Vehicle v5 = new Car(8);

Car c1 = new Car(8);

Car c2 = new Car(6);

What value does each of the following method call return? Or indicate if the method call causes an error.

1. v1.isBig(); false
2. v4.isBig(); true because Truck does not have the isBig() method, and will not override as the vehicle class contains the isBig() method
3. t1.isBig(); true

ti variable is part of the truck class and calls the isBig() from the truck class

1. v5.isBig(); true, as both vehicle and car have the isBig() method, though at run-time the program will run isBig(); from the car class as v5 stores a car object
2. c1.isBig(); true, as c1 is car, car has isBig() method

**Following methods are overloading**

1. v1.equals(v2); true

true, v1 variable points to vehicle class, and v2 fits the parameter, compares wheels

1. v1.equals(v3); false

false, similar to n, v1 points to vehicle class and uses vehicle class .equals(Vehicle other) method

1. v1.equals(t1); false

false, as v1 points to a vehicle class, and t1 is a vehicle (a truck is a vehicle), so they compare the number of wheels on the two objects

1. v1.equals(c1); false

false, as v1 points to vehicle class, and c1 is a car (a car is a vehicle), so they compare wheels

1. v3.equals(c1); false

similar to before, compares wheels.

1. v4.equals(v2); false

Although v4 contains a truck object, at compile (overloading happens at compile time), this uses the .equals method in the vehicle class, and both are vehicles and compares wheels

1. v4.equals(t1); true

ALthough v4 contains a truck object, at compile time, v4 is a vehicle, and uses the vehicle .equals method. Additionally, by casting the v4 to a truck, which it is, (which would compile as java does not look at what v4 is), (Truck(v4)).equals(t1), it would use the truck equals method and compare load capacity

1. t1.equals(v3); false

Initially points to the truck class as t1 points to the truck class, but goes to the parent class as v3 is not necessarily a truck (if it is, which it isn’t, it would need an explicit cast). Therefore this method uses the vehicle class method

1. t1.equals(v4); false

Similiar to u), except v4 does contain a truck object. However, at compile time v4 is considered a vehicle object and compares wheels. An explicit cast is needed to consider v4 a truck.

1. t1.equals(t2); false

points to truck class because of t1 variable type (truck), and runs the .equals method from the truck class because t2 is also type truck

1. t1.equals(c1); false

points to the truck class but cannot compare as the variable of c1 is not a truck, so it goes to the parent class which satisfies the condition that c1 and t1 are both vehicles and therefore compares wheels

1. c1.equals(v1); false

Checks car class for a .equals method which does not exist, so it goes to the vehicle class to compare wheels

1. c1.equals(t1); false

checks car class for a .equals method which does not exist, so it goes to the vehicle class to compare wheels. In the hierarchy both t1 and c1 have the parent class of vehicle (although are not directly related). Therefore, they are compared by their wheels