```python
from functools import cmp_to_key
from Identification import Identification

# Drone.py
# Authors: Diogo RosĂĄrio, JoĂŁo Raposo
# Description: Represents the master agent object and conducts all necessary computations to provide the larger identifications among the three drones
# - drone_A: Drone A
# - drone_B: Drone B
# - drone_C: Drone C
# - count_A_cars: Integer value of the number of cars identified by drone A
# - count_A_Houses: Integer value of the number of houses identified by drone A
# - count_A_Trees: Integer value of the number of trees identified by drone A
# - count_B_cars: Integer value of the number of cars identified by drone B
# - count_B_Houses: Integer value of the number of houses identified by drone B
# - count_B_Trees: Integer value of the number of trees identified by drone B
# - count_C_cars: Integer value of the number of cars identified by drone C
# - count_C_Houses: Integer value of the number of houses identified by drone C
# - count_C_Trees: Integer value of the number of trees identified by drone C
# - identifications: List of the result identifications
class Master():

    def __init__(self, droneA, droneB, droneC):
        self.drone_A = droneA
        self.drone_B = droneB
        self.drone_C = droneC

        self.count_A_cars = 0
        self.count_A_Houses = 0
        self.count_A_trees = 0

        self.count_B_cars = 0
        self.count_B_Houses = 0
        self.count_B_trees = 0

        self.count_C_cars = 0
        self.count_C_Houses = 0
        self.count_C_trees = 0


        self.identifications = self.calculateIdentifications(self.drone_A, self.drone_B, self.drone_C)
        for ident in self.identifications:
            if(ident.drone == "A"):
                if(ident.class_type == 'Car'):
                     self.count_A_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_A_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_A_trees += 1

            elif(ident.drone == "B"):
                if(ident.class_type == 'Car'):
                     self.count_B_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_B_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_B_trees += 1

            elif(ident.drone == "C"):
                if(ident.class_type == 'Car'):
                     self.count_C_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_C_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_C_trees += 1

    #Calulate the identifications that are bigger than others. Check the method "comparator" in identifications.py to see what it means "Bigger"
    def calculateIdentifications(self, drone_A, drone_B, drone_C):
        identificationsCheckpoint = []

        for drone_A_ident in drone_A.identifications:
            self.addIdentification(identificationsCheckpoint, drone_A_ident)

        for drone_B_ident in drone_B.identifications:
            self.addIdentification(identificationsCheckpoint, drone_B_ident)

        for drone_C_ident in drone_C.identifications:
            self.addIdentification(identificationsCheckpoint,drone_C_ident)

        return identificationsCheckpoint

    # Tries to add 1 identification.
    # If doesnt collide with another identification -> adds to the list of identifications
    # If does collide with another identification(s) performs the method "resolveCollision"
    def addIdentification(self, list,box2):

        if(box2.drone_confidence * box2.confidence != 0):
            if(self.canCollide(list,box2)):
                indexes = self.findAllThatCollide(list, box2)
                self.resolveCollision(list, indexes, box2)
            else:
                list.append(box2)


    # Identifies all indices of identifications Y with which a given identification X collides.
    # Returns the list of indices
    def findAllThatCollide(self, list, box2):
        indexs = []
        for box1 in list:
            if(box1.checkCollision(box2)):
                indexs.append(list.index(box1))

        return indexs

    # Verifies whether Identification X collides with at least one identification Y from the provided list.
    # Returns true if so.
    def canCollide(self, list, box2):
        for box1 in list:
```

```python
        for box1 in list:
            if(box1.checkCollision(box2)):
                return True
        return False

    # In the event of a collision, executes a sequence of methods to eliminate the collision.
    def resolveCollision(self, list, indexes, box2):
        boxes = []

        # Get all the boxes from the indeces
        for i in indexes:
            boxes.append(list[i])

        # Check if box2 has lower confidence that ANY other box that collide with
        # If yes, dont add
        if(self.hasLowerConfidence(boxes, box2)):
            return

        # If last step doesnt work, find the box with lowest confidence and remove it and test
        boxes = sorted(boxes, key=cmp_to_key(Identification.comparator))
        for box1 in boxes:
            if(Identification.comparator(box2, box1) >= 0):
                index = list.index(box1)
                list.pop(index)
                if(not self.canCollide(list, box2)):
                    list.append(box2)
                    return

    # Returns true if at least one identification (box1) from the given list is larger than identification Y (box2).
    def hasLowerConfidence(self, boxes, box2):
        for box1 in boxes:
            if(Identification.comparator(box1, box2) >= 0):
                return True
        return False
```