BUILD A CAR GIVEN ALL TASKS
AND EACH TASK'S DEPENDENCIES

BUILD A CAR GIVEN ALL TASKS AND EACH TASK'S DEPENDENCIES

SET UP THE DATA STRUCTURE THAT YOU PLAN TO USE TO STORE A TASKS AND ITS DEPENDENCIES

DEPENDENCIES OF TASK "A" ARE ALL TASKS WHICH SHOULD BE COMPLETE BEFORE "A" CAN EXECUTE

"PAINT A CAR" DEPENDS ON "BUILD CHASSIS OF THE CAR", THE CHASSIS NEEDS TO BE READY BEFORE IT CAN BE PAINTED

THE METHOD SHOULD TAKE IN A LIST OF ALL TASKS WHICH ARE NEEDED TO BUILD A CAR ALONG WITH THEIR DEPENDENCIES AND EXECUTE EVERY TASK IN ORDER TO BUILD THE CAR

THE TASKS ARE IN ANY ORDER, IT'S UP TO YOU TO DETERMINE SOME ORDER IN WHICH IT CAN BE EXECUTED

TASKS AND DEPENDENCIES

SAY THAT ALL THE TASKS NEEDED TO BUILD A CAR ARE:

A, B, C, D, E, F, G, H

LET'S SAY THE DEPENDENCIES ARE:

B DEPENDS ON A
D DEPENDS ON E
C DEPENDS ON A, B, D
F DEPENDS ON C

AN ACCEPTABLE ORDER OF PERFORMING THE TASKS ARE:

E->D->A->B->C->F

OR

A->E->B->D->C->F

REMEMBER THAT YOU CAN'T EXECUTE A TASK UNLESS IT'S DEPENDENCIES HAVE COMPLETED

STORE THE DEPENDENCIES SUCH THAT THEY ARE EASILY ACCESSIBLE FROM A TASK

RECURSIVELY EXECUTE THE DEPENDENCIES TILL YOU GET TO THE TASK

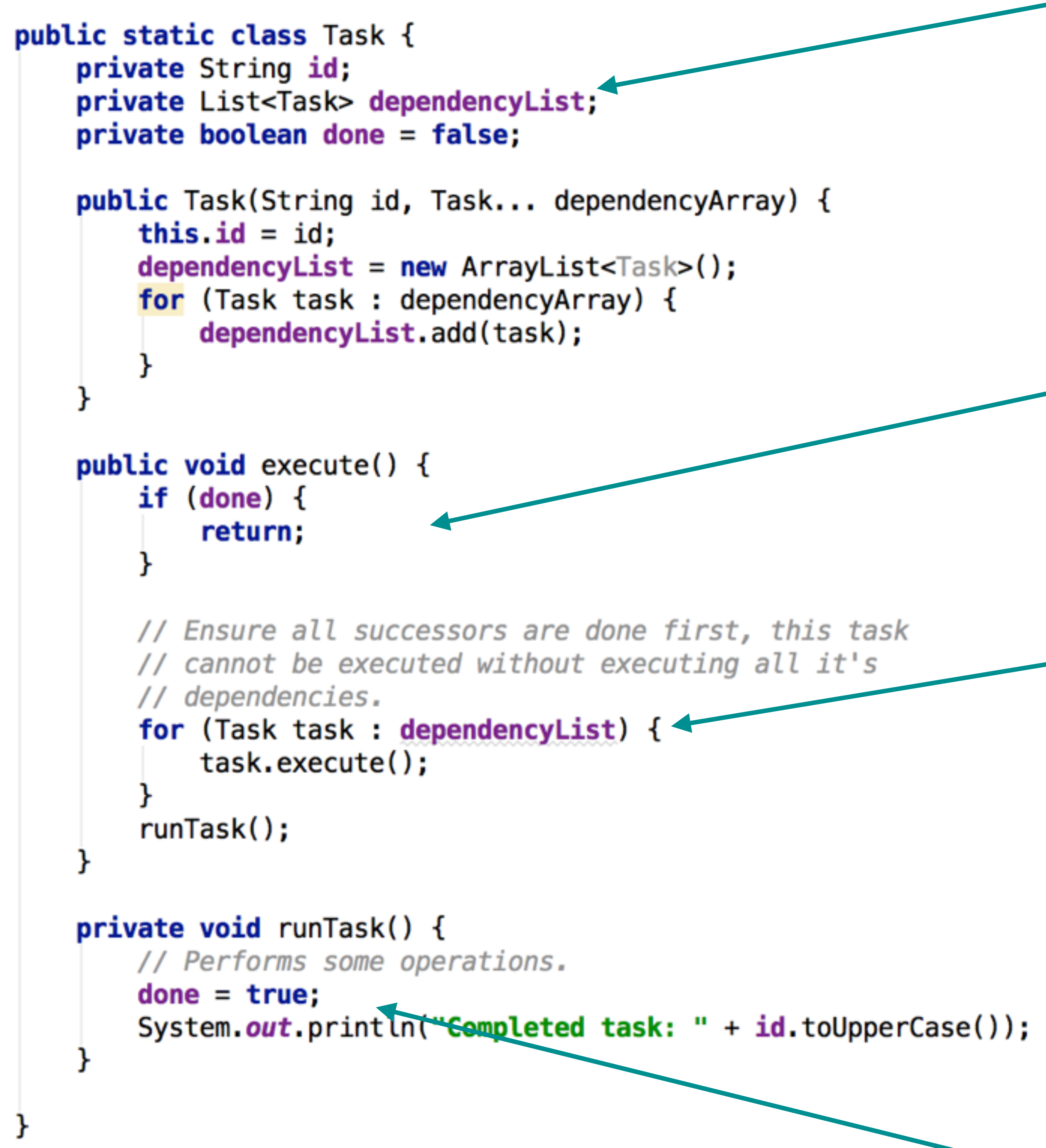
WHAT IS THE BASE CASE?

THE CURRENT TASK HAS BEEN EXECUTED, IT'S MARKED DONE

WHAT IS THE RECURSIVE CASE?

EXECUTE DEPENDENCIES BEFORE COMING TO THE CURRENT TASK

A SINGLE TASK



```java
public static class Task {
    private String id;
    private List<Task> dependencyList;
    private boolean done = false;

    public Task(String id, Task... dependencyArray) {
        this.id = id;
        dependencyList = new ArrayList<Task>();
        for (Task task : dependencyArray) {
            dependencyList.add(task);
        }
    }

    public void execute() {
        if (done) {
            return;
        }

        // Ensure all successors are done first, this task
        // cannot be executed without executing all it's
        // dependencies.
        for (Task task : dependencyList) {
            task.execute();
        }
        runTask();
    }

    private void runTask() {
        // Performs some operations.
        done = true;
        System.out.println("Completed task: " + id.toUpperCase());
    }

}
```

HOLDS A LIST OF DEPENDENCIES OR THE TASKS WHICH HAVE TO COME BEFORE THE CURRENT TASK

IF THE TASK IS DONE, JUST RETURN

BEFORE EXECUTING THE CURRENT TASK, RECURSIVELY CALL EXECUTE ON ALL THE DEPENDENCIES OF THIS TASK

SIMPLY MARK THE CURRENT TASK AS DONE

BUILD A CAR

BUILD A CAR IS NOW VERY SIMPLE, ALL THE COMPLEXITY IS HIDDEN IN THE EXECUTE() METHOD OF THE TASK

```java
public static void buildCar(List<Task> taskList) {
    for (Task task : taskList) {
        task.execute();
    }
}
```

JUST ITERATE THROUGH EVERY TASK AND CALL EXECUTE() ON IT

THERE MIGHT BE SOME TASKS, WHICH STAND ALONE, NOTHING DEPENDS ON THEM

THE TASKS AND IT'S DEPENDENCIES FORM A DIRECTED ACYCLIC GRAPH

THE GRAPH MAY NOT BE FULLY CONNECTED, I.E. CERTAIN TASKS STAND COMPLETELY ALONE

WE HAVE TO VISIT EVERY TASK TO EXECUTE IT, THE COMPLEXITY IS O(N), WHERE N IS THE NUMBER OF TASKS