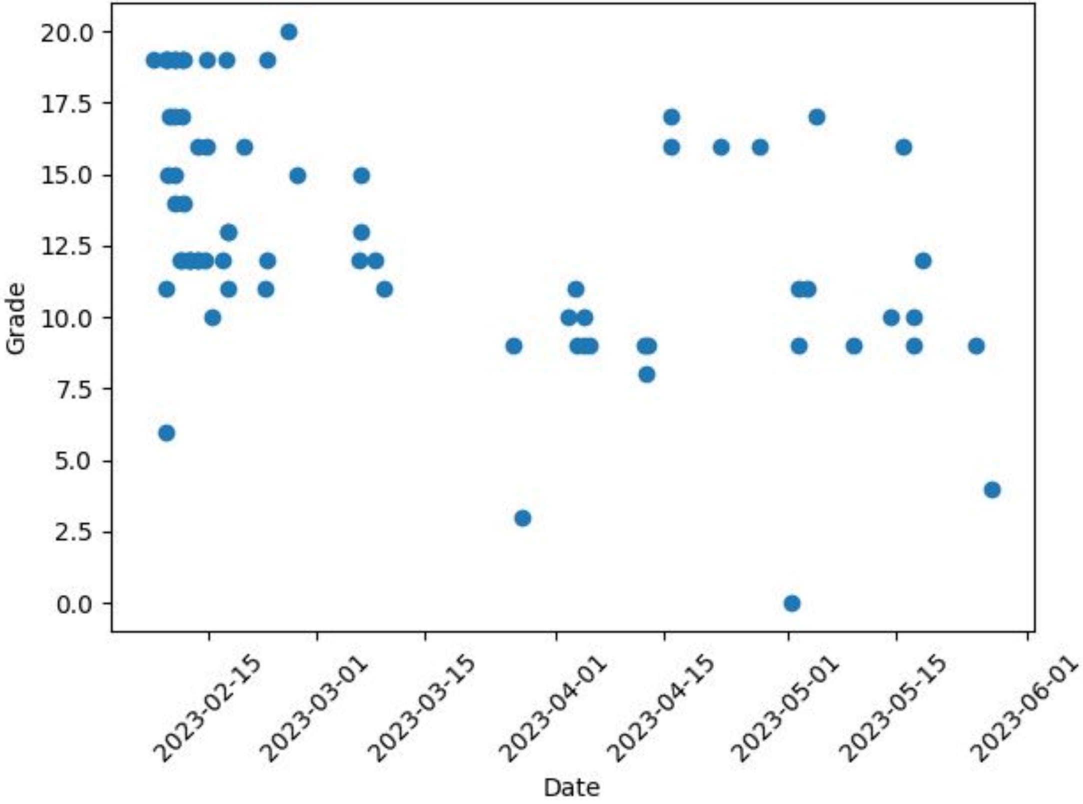
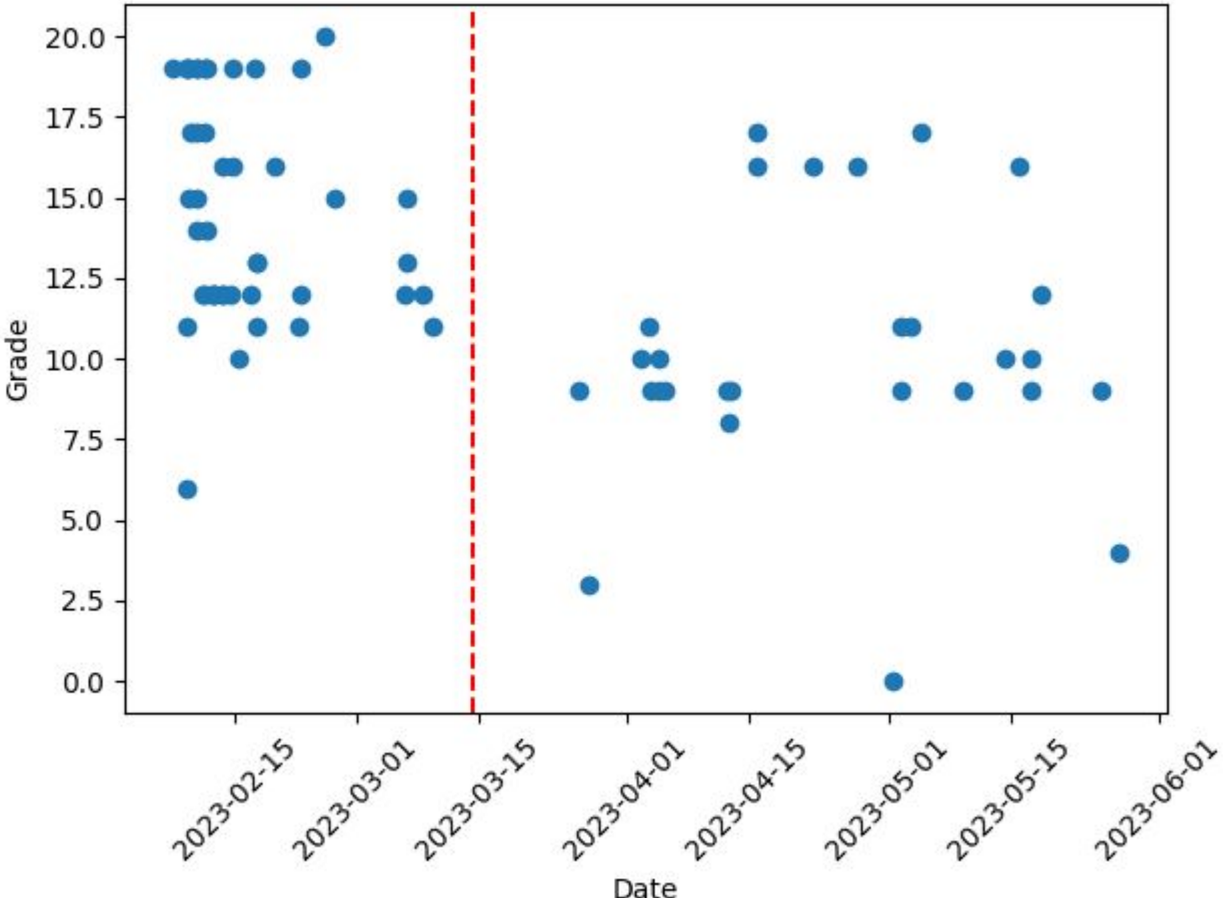
# CP project 2025

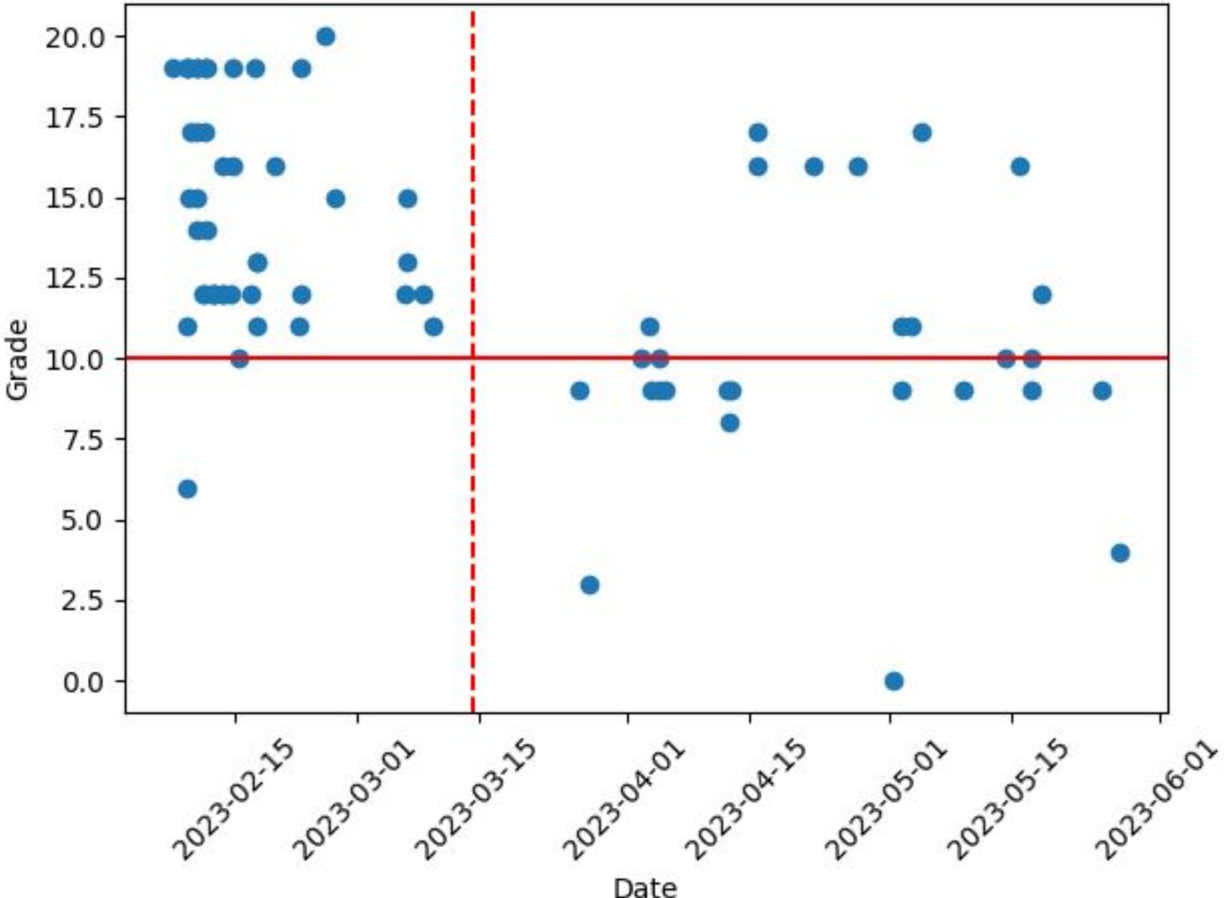
# Grades in 2023, depending on date of 2nd commit



# Grades in 2023, depending on date of 2nd commit

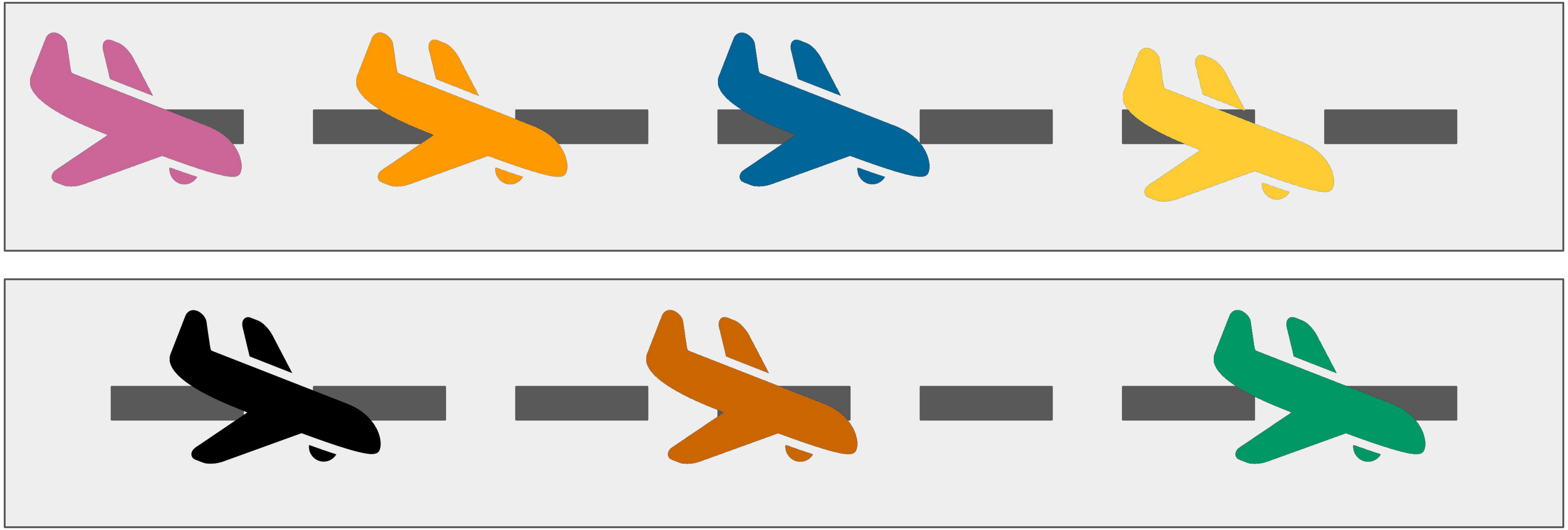


# Grades in 2023, depending on date of 2nd commit



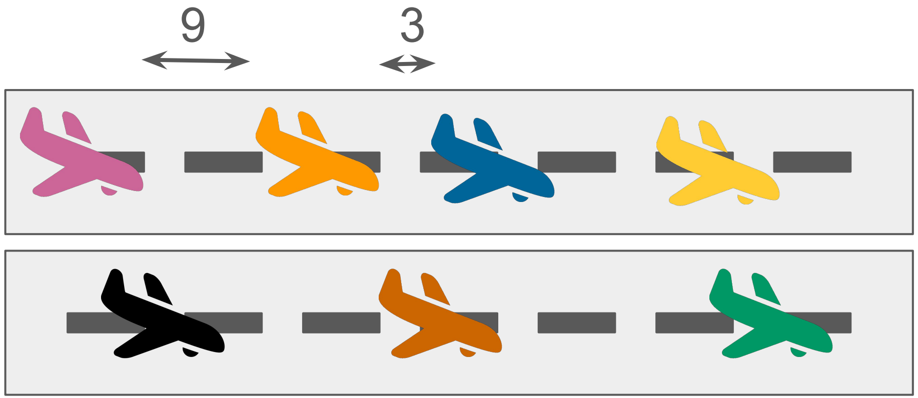
# Schedule plane landing

# Schedule plane landing



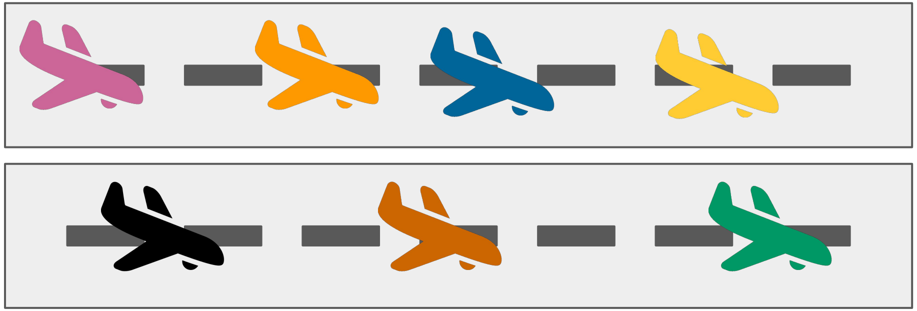
# List of constraints

- Limited number of lanes for landing
- Each plane has
  - A deadline for landing
  - An associated type
- There is a delay between 2 landing of planes
  - Delay dependent of the plane type



| Delay | 🟪 | 🟧 | 🟦 |
|---|---|---|---|
| 🟪 | 5 | 9 | 7 |
| 🟧 | 10 | 9 | 3 |
| 🟦 | 8 | 6 | 5 |

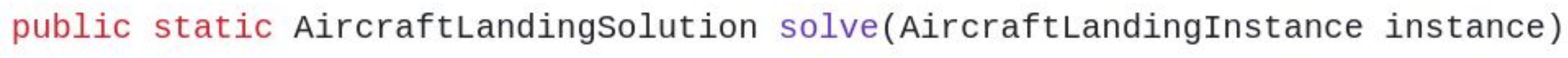
# List of constraints



- Limited number of lanes for landing
- Each plane has
  - A deadline *d* for landing
  - An associated type
  - **A preferred landing time $t^*$**
    - $t^* <= d$
    - Landing a plane *p* at time *t* induces a $\mathbf{cost_p = |t^* - t|}$
- There is a delay between 2 landing of planes
  - Delay dependent of the plane type
- Objective: **minimize sum of cost**

# In terms of code

- Go to the AircraftLanding class (package minicp.examples), meant to encode the problem
- We provide
  - Instance parsing
  - Solution class with checker
  - Unit tests for checking model
- Implement the function solve



```
public static AircraftLandingSolution solve(AircraftLandingInstance instance)
```

- This function must return your best found solution to the problem
- Use your solver to model the problem and solve it!

# Grading

- You need to solve 4 instances
- Your solver is run during **3 CPU minutes** and must return the best solution found within this time budget
- The grade for each instance is as follow
  1. Finding a **feasible** solution: **7/20**
  2. Passing a **first** optimisation **threshold** for an instance: **12/20**
  3. Passing a **second** optimisation threshold for an instance: **at least 15/20**
     - The best solution found awards 20/20
     - The worst solution found (better than the 2nd threshold) awards 15/20
     - Linear in between
- Grade for the project = mean between the 4 instances

# Example

Instance X

- 1st optimisation threshold: 1000
- 2nd optimisation threshold: 500

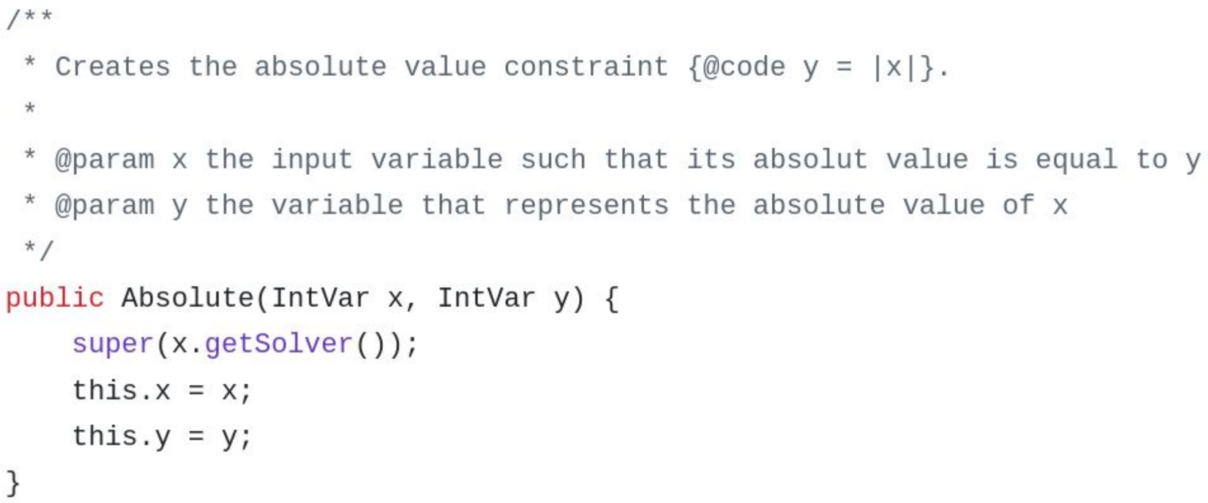
6 submissions:

| Student | Solution value | Grade |
|---|---|---|
| A | No solution | 0/20 |
| B | 1500 | 7/20 |
| C | 700 | 12/20 |
| D | 400 | At least 15/20 |
| E | 350 | At least 15/20 |
| F | 360 | At least 15/20 |

- E get 20/20 (best solution),
- D get 15/20 (“worse” solution being better than 2nd threshold),
- F gets 17.5/20 (linear in between)

# Tips and tricks

- It is ***strongly advised*** to implement the Absolute constraint
  - Useful to model the landing cost $\mathbf{cost}_p = |t^* - t|$
  - Unit tests for this constraint are provided



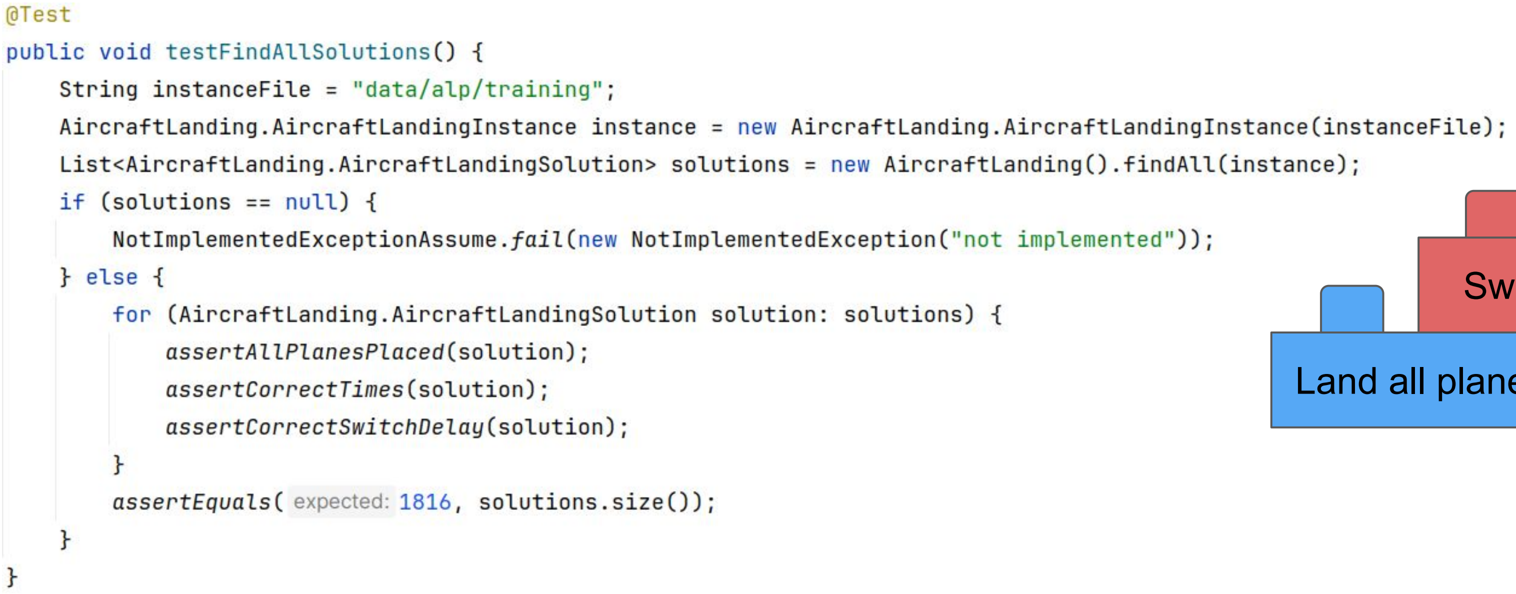
```java
/**
 * Creates the absolute value constraint {@code y = |x|}.
 *
 * @param x the input variable such that its absolut value is equal to y
 * @param y the variable that represents the absolute value of x
 */
public Absolute(IntVar x, IntVar y) {
    super(x.getSolver());
    this.x = x;
    this.y = y;
}
```

# Tips and tricks (2)

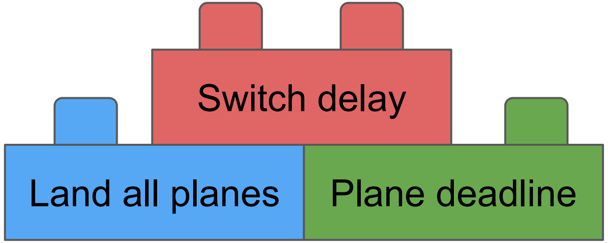
- CP = Model + Search
  - **Dedicate time on both**: a good model with a bad search will not find a solution to the instances
- Check that your model is correct before spending time on search
  - The solution class AircraftLandingSolution contains a checker. Creating an invalid solution throws an error.
  - A function that you *can* implement (and is not evaluated) is provided
    public static List<AircraftLandingSolution> findAll(...)
    And is unit tested (the tests do not count in the grade)

# Tips and tricks (3)

- Check that your model is correct before spending time on search
  - Try to compose your model gradually, constraint after constraint



```
@Test
public void testFindAllSolutions() {
    String instanceFile = "data/alp/training";
    AircraftLanding.AircraftLandingInstance instance = new AircraftLanding.AircraftLandingInstance(instanceFile);
    List<AircraftLanding.AircraftLandingSolution> solutions = new AircraftLanding().findAll(instance);
    if (solutions == null) {
        NotImplementedExceptionAssume.fail(new NotImplementedException("not implemented"));
    } else {
        for (AircraftLanding.AircraftLandingSolution solution: solutions) {
            assertAllPlanesPlaced(solution);
            assertCorrectTimes(solution);
            assertCorrectSwitchDelay(solution);
        }
        assertEquals( expected: 1816, solutions.size());
    }
}
```



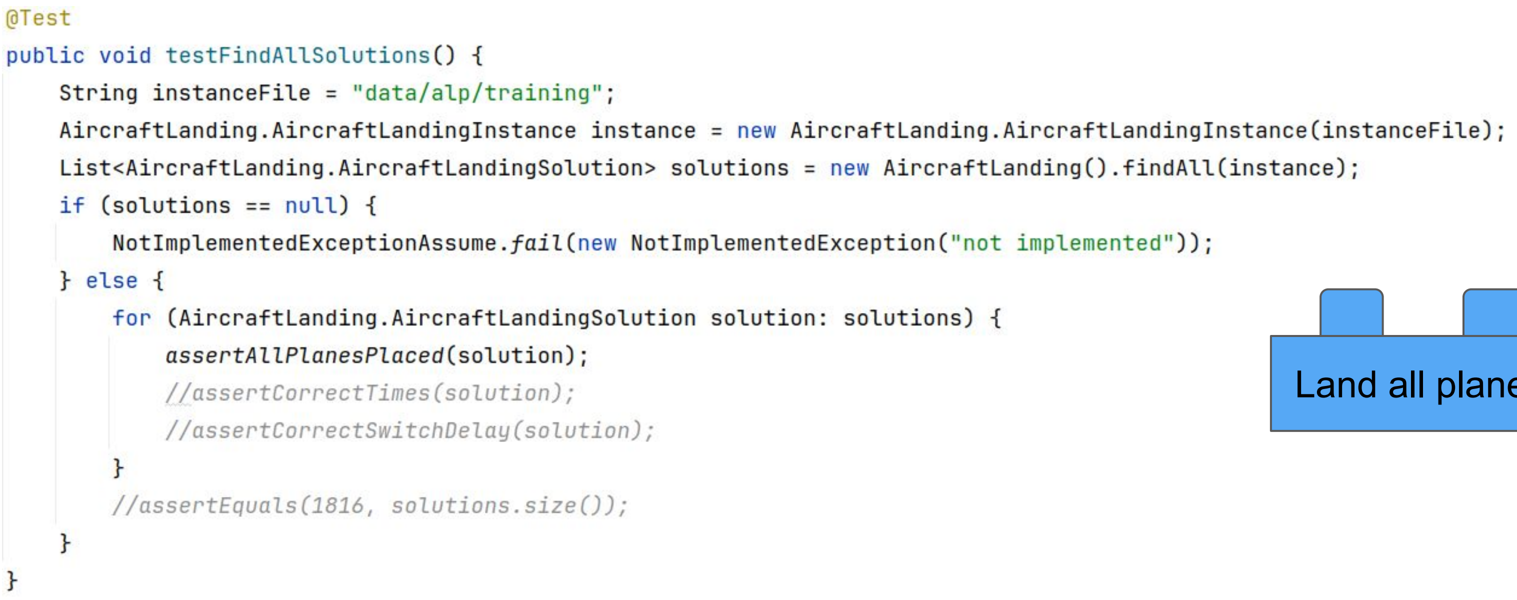
Switch delay

Land all planes

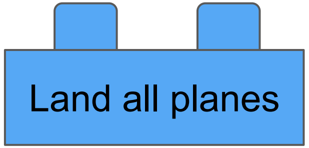
Plane deadline

# Tips and tricks (3)

- Check that your model is correct before spending time on search
    - Try to compose your model gradually, constraint after constraint

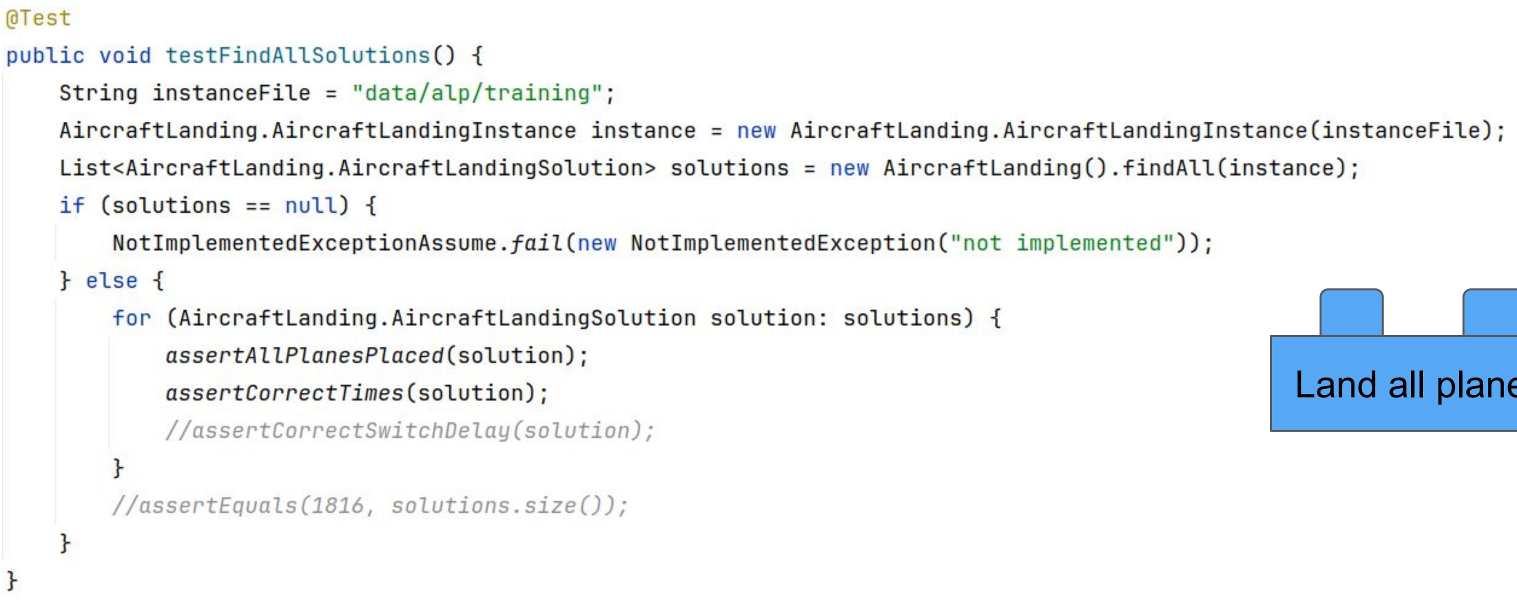


```
@Test
public void testFindAllSolutions() {
    String instanceFile = "data/alp/training";
    AircraftLanding.AircraftLandingInstance instance = new AircraftLanding.AircraftLandingInstance(instanceFile);
    List<AircraftLanding.AircraftLandingSolution> solutions = new AircraftLanding().findAll(instance);
    if (solutions == null) {
        NotImplementedExceptionAssume.fail(new NotImplementedException("not implemented"));
    } else {
        for (AircraftLanding.AircraftLandingSolution solution: solutions) {
            assertAllPlanesPlaced(solution);
            //assertCorrectTimes(solution);
            //assertCorrectSwitchDelay(solution);
        }
        //assertEquals(1816, solutions.size());
    }
}
```

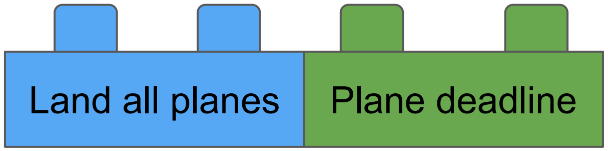


# Tips and tricks (3)

- Check that your model is correct before spending time on search
  - Try to compose your model gradually, constraint after constraint

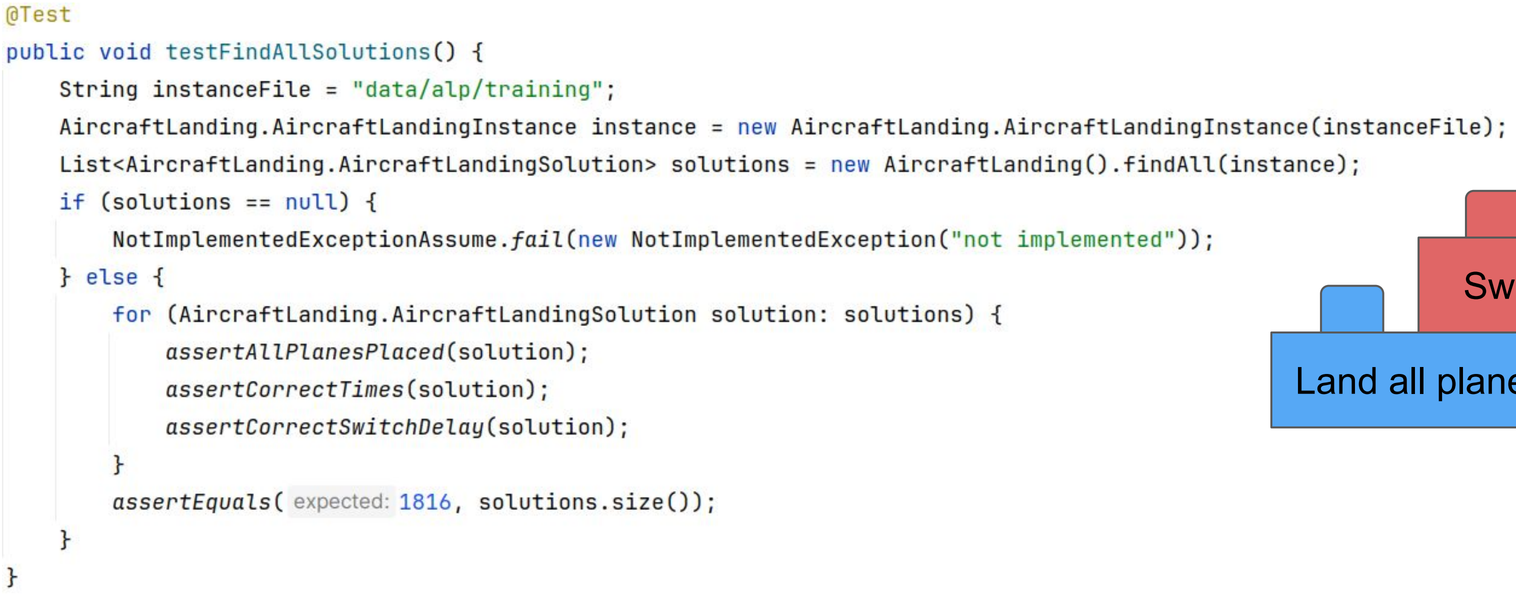


```java
@Test
public void testFindAllSolutions() {
    String instanceFile = "data/alp/training";
    AircraftLanding.AircraftLandingInstance instance = new AircraftLanding.AircraftLandingInstance(instanceFile);
    List<AircraftLanding.AircraftLandingSolution> solutions = new AircraftLanding().findAll(instance);
    if (solutions == null) {
        NotImplementedExceptionAssume.fail(new NotImplementedException("not implemented"));
    } else {
        for (AircraftLanding.AircraftLandingSolution solution: solutions) {
            assertAllPlanesPlaced(solution);
            assertCorrectTimes(solution);
            //assertCorrectSwitchDelay(solution);
        }
        //assertEquals(1816, solutions.size());
    }
}
```

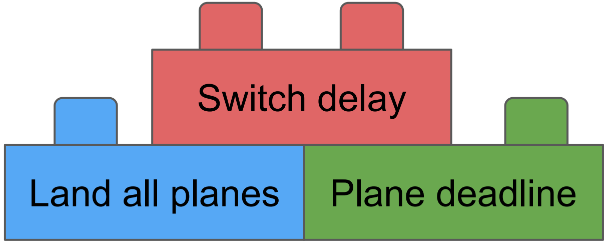


# Tips and tricks (3)

- Check that your model is correct before spending time on search
    - Try to compose your model gradually, constraint after constraint



```
@Test
public void testFindAllSolutions() {
    String instanceFile = "data/alp/training";
    AircraftLanding.AircraftLandingInstance instance = new AircraftLanding.AircraftLandingInstance(instanceFile);
    List<AircraftLanding.AircraftLandingSolution> solutions = new AircraftLanding().findAll(instance);
    if (solutions == null) {
        NotImplementedExceptionAssume.fail(new NotImplementedException("not implemented"));
    } else {
        for (AircraftLanding.AircraftLandingSolution solution: solutions) {
            assertAllPlanesPlaced(solution);
            assertCorrectTimes(solution);
            assertCorrectSwitchDelay(solution);
        }
        assertEquals( expected: 1816, solutions.size());
    }
}
```



# Tips and tricks (4)

- If you use some randomness during the search, be sure to use **seeds** for RNG.
  - Ensure that a good solution found on your laptop can also be retrieved on inginious
- Inginious will in most cases be slower than your laptop
  - If you find your first solution on your high-end computer at 2min57 but inginious does not find it in 3 minutes, giving you 0/20, you should work on your solver, not complain on inginious
  - ***All thresholds can be reached on inginious with a good approach***
- Dedicate enough time for implementing the search
  - If you have a working model but no efficient search procedure 1 week before the deadline, you are in trouble

# Tips and tricks (5)

- **Don’t cheat.**
  - Forbidden to use code from other students, being this year or previous years.
- We run anti plagiarism tools
  - on your submissions
  - And on submissions from previous years
- In past editions of the course, we caught students that plagiarized each others
  - And it did not end well for them
  - Renaming of variables and changing for loops into while loops won’t fool the system
- After the end of the project, we might pick students to come and explain their code

# Some Commonly Asked Questions

- *Can I implement custom constraints?*
  - Yes
- *Can I use different values of parameters / search procedure depending on the instance?*
  - Yes
- *Should I use LNS (cf module 6)?*
  - Oh yes you should!
- *Can I hard-code a solution into my code?*
  - No
- *Can I share some code with other students?*
  - No
  - But you can of course discuss strategies and approaches

# Deadlines

- Today (14/03/2025):
  - Project starts
  - Moodle forum for the project opens (again, don’t share code there)
- Next week: Inginious tasks for the project open
- End of semester (17/05) @ 23h59: project ends
- Last questions on the project answered: 10/05 @ 23h59
  - Questions asked AFTER this date and time will not be answered
  - Unless for a ***technical issue*** (i.e. you got a 404 when logging in to inginious, the server went on fire, …).
  - **Definitely not** “my submission that I did not compile does not run on inginious”
- Although you have almost 2 months for the project, **don’t start at the last minute**
  - This project takes times!

# Best of luck for the project!

Questions?