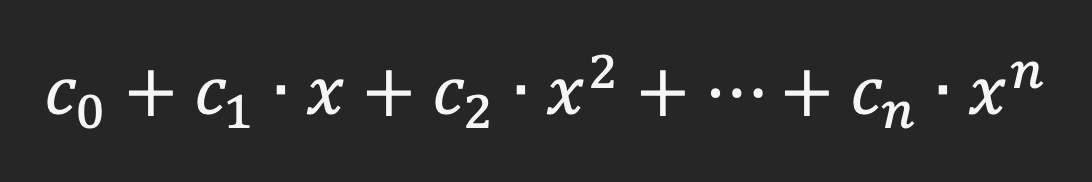
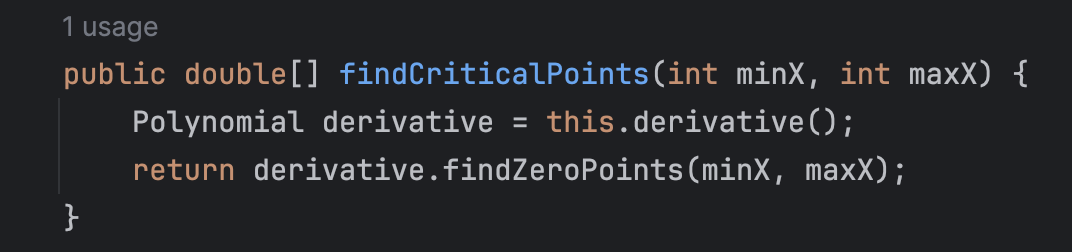
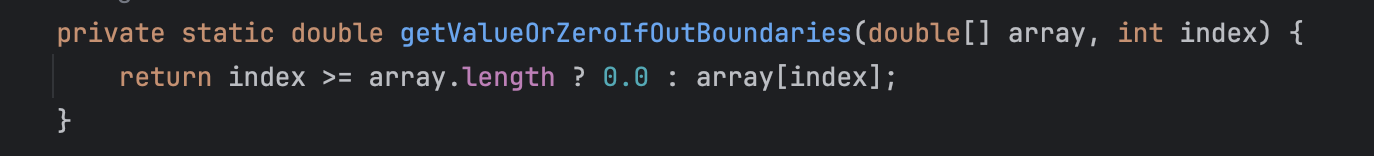
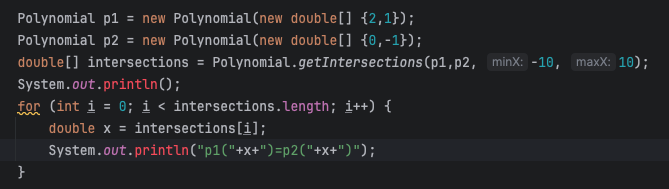
**Workshop 8 - Objects**

Start with the polynomial exercise to **present the idea of creating a logical unit**. (30 min)

Before we dive into creating more complex objects, polynomials at these exercises can be described by coefficient array only and it might be more simple to not use many properties of an object, but getting focused on the purpose of creating an object that has its own behaviors.

Share the skeleton and main files from [here](https://drive.google.com/drive/folders/1YuDZ-MR6AZtZsm95ZJNZqg0Dsg9f0TY9)

1. **Main:**  
   Before letting students go implement the class, use main calls to make a conversation about the way we want to use Polynomial.  
   - How do we create a new instance? (using ‘new’ syntax)  
   - What is stored in the `Polynomial` variable? (a reference - not value)  
   - How do we access `Polynomial` methods? (.evaluate(x) \ .derriviate() \ etc…)  
   - Shortly explain about `toString()`  
   - Go over all methods should be implemented (constructor \ evaluate \ derivative \ findCrtiticalPoints - *integral \ calculateArea are extra and not part of this WS*)
2. **Constructor:**
   1. Briefly explain the usage of coefficients array 
   2. Explain to students that constructor also might be used for validating data in such a way we do not always store the given data as it is given.  
      At this exercise we want to clean the rightest part of the array where it all zeros - as in comments:  
      // coefficients = [3, 1, 0, 0] <=> 3 + X <=> [3, 1]
3. **Evaluate - nothing much to explain, just calculating y(x)**
4. **Derivative:**
   1. Briefly explain how to calculate derivatives of polynomials.
   2. What should be the length of the derivative coefficients array? (n-1)
   3. Use the same object type to represent the derivative (return it as Polynomial)
5. **findCriticalPoints:**
   1. There is a method called `findZeroPoints` which is private - explain briefly about this kind of exposure.
   2. This method should be use over the derivative polynomial in order to find all critical points - let students discuss either it can be used or not (since it’s not directly get called from it might be confusing)
6. **getIntersections: (not in skeleton nor main)**This method is not in skeleton so you can discuss with students how they should implement methods like that.  
   Since this method has nothing to deal with a specific polynomial it might be better to use it as a **static method**.  
   Basically you should guide the students to use the existing structure of the polynomial so it can be reused here to find intersection (create a polynomial of reducing one polynomial by the other, then the result will be all zero points of the new polynomial).  
   **[Pay attention]** Since the Polynomials might be from a different degree, you might use a private static method to return default 0 for indices out of boundaries - lead the students to deal with a variety of method signatures during the object methods.  
     
   **[Finally]** ask students to call this method from main, how should they approach static methods?  
   

DailyCalendar: (40 min)

In this section we’ll **focus on creating the right properties** to support required behaviors.

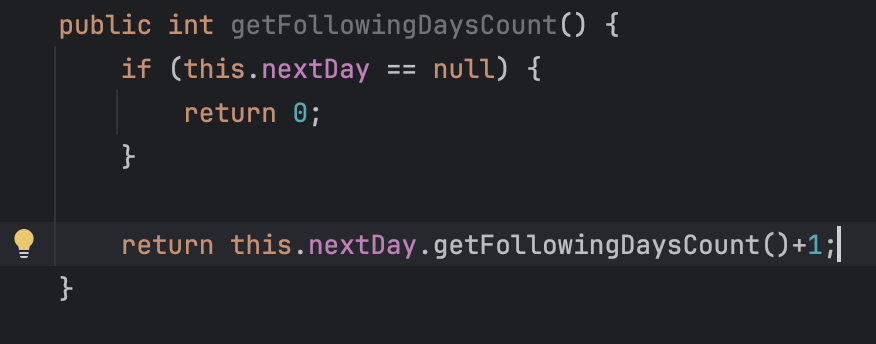
Share the skeleton and main files from [here](https://drive.google.com/drive/folders/1xYAQ5lMgDfjZSMy5nklbnUCq0fy7Sise)

Start with presenting all required methods used in main.

Let the students note that there are 2 different constructors, one that takes as integers day, month and year - and the second takes it as string (both takes hourSplits - which means how much plans can be stored in one hour).

After a short brief on main, go over DailyCalendar skeleton, present the methods of getAlignedDay, getAlignedMonth, isLeapYear.  
These methods are static and they use the static property numbersOfDayEachMonth.  
Explain about static properties and how they act in objects.

Discuss about these requirements, what properties should we store in our object.

1. **Constructors:**  
   There are 2 constructors to implement, you might guide your students from the beginning to create an initialize method to be used from both constructors or let them build the first constructor then extract it to initialize method so it can be used for the second constructor as well.  
   Explain the usage of different constructors for the same object.  
   Students should use the align methods to store valid properties.  
   **setPlanAt:**Guide students use getPlanIndex in order to map the array and find relevant slots to the required hour.  
   The plan should be stored in the first slot found or not stored at all if all slots in use.
2. **toString: nothing much to explain**
3. **getTotalDailyCalendarsCreated: (not in skeleton)**Ask students to create a static method that returns the count of all DailyCalendars created.  
   Explain more about usage of static properties.  
   Pay attention to the counter to be increased via initialization.  
   Why should we define this static member as private? (can be dangerously changed)
4. **getNextDayCalendar:**Here we start building motivation to linked lists - do not mention that, but let the students understand the concept of storing as member another object.  
   At this method, students implement lazy creation (when this.nextDay == null) of nextDay and store it as member then return it.
5. **getFollowingDaysCount: (not in skeleton)**Ask the students to implement a recursive method to find the count of next days that chained.  
   
6. **getNextNDaysString:**Ask students to implement a method that return a string that merging results of toString of the next n days or to the last next day found.

If left time, go over to Polynomials to implement integral & calculateArea.