**Suggested by ChatGPT (after prompts)**

Here’s a structured idea for a **single semester-long assignment** broken into **4–5 stages**, each aligned with your weekly content. The assignment grows naturally from simple classes and data types into a system with primitive arrays and methods. All stages use **BlueJ-friendly design**, and early stages can be supported with paper-based design tasks (e.g., UML diagrams, trace tables).

**📘 Could be used as a running example**

**Semester Project: Student Record Management System**

A simple system to manage and report on student data: name, ID, scores, etc.

**🌱 Stage 1 (Weeks 2–3): Design and Basic Class Skeletons**

**Theme:** *Getting started with class structure and basic data representation.*

* Students complete a **written task**:
  + Design a Student class with fields like name, id, and mark.
  + Describe types for each field and write out basic method headers (e.g., getName(), setMark(int)).
  + Draw a **UML diagram** by hand or using a simple tool.
* Deliverable: A written document + partially coded BlueJ class stubs (no logic yet).

**🧩 Stage 2 (Weeks 4–5): Implementing Logic and Control Flow**

**Theme:** *Adding methods and control logic.*

* Implement the Student class fully:
  + Add constructors, getters, and setters.
  + Add methods like isPassing() (e.g., returns true if mark >= 50)
* Include a StudentTester class with main() to interact with Student objects.
* Encourage console I/O or use the object bench for interaction.

**🔄 Stage 3 (Weeks 6–7): Modularity with Methods**

**Theme:** *Enhancing reusability and structure.*

* Create a StudentUtils class with static helper methods:
  + printStudentDetails(Student s)
  + updateMark(Student s, int newMark)
* Optional paper exercise: trace method calls and predict output.

**🧮 Stage 4 (Weeks 8–9): Arrays and Reporting**

**Theme:** *Introducing arrays and simple processing.*

* Add a new StudentManager class to store **an array of Students**.
  + Method to add a Student to the list.
  + Method to print all students.
  + Method to calculate class average.
  + Use a fixed-size array (e.g., 10 students).
* Students test via a menu system or object bench.

**🎓 Stage 5 (Weeks 10–11): Extension / Optional Challenge**

**Theme:** *Consolidation and stretch goal.*

* Add:
  + Method to find the top-performing student.
  + A search method (findStudentById(String id)).
  + Maybe an export-like feature: getStudentListString() returns all students as a formatted string.
* Final delivery: Full BlueJ project + short reflection on what they learned or found challenging.

**✍️ Support Ideas**

* **Paper-based activities** early on:
  + UML diagrams
  + Manual dry runs (trace tables for loop logic or array processing)
  + Code reading & completion
* **Mini labs** that feed into the assignment:
  + Mark calculator → becomes part of Student
  + Simple ArraySum exercise → helps with class average method

**🧾. Could be set as assignment in parallel with the labs above**

**Assignment Title: Student Co-Curricular Engagement Tracker**

**Brief**

You are tasked with building a Java-based system to manage and report on **students' participation in non-academic co-curricular activities**, such as clubs, societies, and volunteering. This extends beyond marks and incorporates data such as hours volunteered, roles held, and categories of activity (e.g., Sport, Arts, STEM, Leadership).

This system is intended for a university's new “Engagement Transcript” initiative that complements academic records.

**Core Requirements**

**Part 1: Class Design (Week 6–7)**

* Design a Student class with at least the following fields:
  + name, studentId, and a list of activities.
* Design an Activity class with fields such as:
  + name, category, hours, and isLeadershipRole (boolean).
* Include constructors, getters/setters, and string representations for both classes.

**Part 2: Object Interaction & Arrays (Week 8–9)**

* Implement a StudentManager class that stores a fixed-size array of Student objects.
* Include methods to:
  + Add a new student.
  + Add an activity to a student (by student ID).
  + Calculate and return total hours volunteered by a student.
  + Generate a list of students who have held leadership roles.

**Part 3: Reporting & Analysis (Week 10–11)**

* Add methods to:
  + List all activities in a specific category (e.g., all ‘STEM’ activities).
  + Identify the student with the highest engagement (e.g., total hours or number of activities).
  + Provide a formatted report of each student's engagement transcript.

**Optional Extensions (for high-performing students)**

* Add inheritance: SportActivity, ArtsActivity (specialized subclasses of Activity with additional fields like teamName or artForm).
* Use polymorphism in generating reports (e.g., activities report differently based on type).
* Implement abstract base class EngagementRecord if expanding to other data types like internships or mentoring.

**Deliverables**

* Fully implemented and tested Java classes.
* A short **developer’s report** (max 2 pages) explaining:
  + Design choices.
  + Testing strategy (can include JUnit).
  + Known limitations.

**Why AI Bots Can't Solve It Easily**

* It's not a common stock problem.
* It requires **interpretation and integration** of multiple concepts (arrays of objects, OOP principles).
* It invites **subjective design decisions** (e.g., how to structure leadership roles).
* Extensions involve **creative use of inheritance and polymorphism**, which bots can't guess without guidance.

Would you like me to write a handout version or prepare starter files for students (e.g., Student.java, partially complete StudentManager.java, etc.)?