Lesson 12: Decomposition and Design

45 minutes

Overview

How do I decide what new methods to write in a program?

Students are introduced to decomposition and top-down design to deconstruct problems into smaller tasks and develop algorithms for these tasks. Students analyze decomposition examples and identify the importance of writing clear and specific pseudocode. Students write and translate algorithms into methods and consider potential edge cases to improve their programs.

Standards

Full Course Alignment

CSA Conceptual Framework

▶ **VAR-1** - To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values

Agenda

Warm Up (10 minutes)
Solving Problems

Activity (30 minutes)
Developing Efficient Solutions
Deconstructing a Problem

Wrap Up (5 minutes)
Software Engineering Skills
Assessment: Check for Understanding

Objectives

Students will be able to:

- Compare programs and identify good vs. poor decomposition
- Identify edge cases for an algorithm
- Write clear and readable code using methods, control structures, and comments

Preparation

- Print copies of the Decomposition handout (one for each pair of students)
- Check the **Teacher's Lounge** for verified teachers on the CSA Forum to find additional strategies or resources shared by fellow teachers

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the students

- **Decomposition** - Handout
- **U1L12 Extra Practice** - Handout

Vocabulary

- **Method Decomposition** - the process of breaking a problem down into smaller parts to write methods for each part
- **edge case** - a bug that occurs at the highest or lowest end of a

range of possible values or in extreme situations
- **redundant code** - code that is unnecessary

Teaching Guide

Warm Up (10 minutes)

Solving Problems

Remarks

> Computer science is all about problem-solving. Solving problems is a core component of a software engineer's work, and we have learned about ways to create and improve solutions in The Neighborhood. Let's consider how we solve problems in our daily lives.

Group: Place students in groups of three or four.

Do This: Have students work with their groups to write an algorithm for the scenario.

Discuss: *What process did you take to decide on your algorithm?*

Discussion Goal: Students share the approaches they took to develop their algorithm. Students discuss how they broke the problem down into smaller tasks, such as sorting the colors into piles and then counting the number of M&M's in each pile.

Activity (30 minutes)

Developing Efficient Solutions (10 minutes)

Remarks

> Sometimes, problems are so big that we need to break them down into manageable tasks first. This process is called decomposition, and we will learn how to use this approach to break down problems.

Do This: Review the lesson objectives.

Do This: Define *decomposition*.

Teaching Tip

Revisit the scenario from the warm up or an example of a personal interest or daily activity. Briefly discuss how the scenario would be broken down into specific tasks or steps.

Do This: Explain the steps to solving a programming problem.

- Understand
- Expected Behavior
- Supporting Information
- Mapping

- Assemble
- Testing

Deconstructing a Problem (20 minutes)

Remarks

> As you work through these problems, use decomposition strategies to break them down into manageable tasks. Consider any edge cases for your solutions and review your solutions for any redundancy.

Group: Place students in pairs.

Distribute: Give each pair a copy of the Decomposition handout.

Do This: Direct students to work with their partners to write pseudocode for one of the choice problems on the Decomposition handout and add their new method to the PainterPlus UML diagram.

Do This: Direct students to Level 3 on Code Studio to complete a choice level to implement their new method.

1 **More PainterPlus Methods**

Remarks

> When software engineers develop solutions, they try to account for potential errors and different ways a user might use the program. However, they have to be careful about being redundant to keep their programs efficient.

Do This: Define *edge case* and *redundant code*.

Remarks

> This is a good time to commit our code and save our new version of the `PainterPlus` class to the Backpack. Anytime we make changes to our programs, it is helpful to commit, or save, our work as a new version in case we need to revert to a previous version.

Do This: Play the music clip to cue committing their code and saving the new version of the `PainterPlus` class to the Backpack.

Remarks

> When we write new code, getting feedback from our peers is helpful to make sure we have met the requirements of the problem efficiently.

Do This: Click through the animated slide to have students participate in the Code Review Call and Response.

Do This: Direct students to complete a code review on their chosen problem on Level 2.

2 **Code Review: More PainterPlus Methods**

Wrap Up (5 minutes)

Software Engineering Skills

Remarks

> You have made a lot of progress developing your software engineering skills. Let's take a moment to reflect on our progress and growth.

Do This: Direct students to use the sentence starters on the Unit 1 Guide to reflect on their progress and growth as software engineers.

Do This: Review the concepts covered in this lesson.

Display: Key Vocabulary

Assessment: Check for Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. You can use these questions as an exit ticket.

Check for Understanding

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes **contact us**.