Lesson 4: Navigating and Painting

45 minutes

Overview

What is a method?

Students learn the syntax for calling methods in Java to practice navigating and painting The Neighborhood. Students call `void` methods from the `Painter` class and practice debugging syntax errors.

Standards

Full Course Alignment

CSA Conceptual Framework

▶ **MOD-1** - Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence

▶ **MOD-2** - Programmers use code to represent a physical object or nonphysical concept, real or imagined, by defining a class based on the attirbutes and/or behaviors of the object or concept

▶ **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)
- **Giving Instructions**

Activity (30 minutes)
- **Calling Methods in Java**
- **Using the Painter Class**

Wrap Up (5 minutes)
- **Finding Errors**
- **Assessment: Check for Understanding**
- **AP Classroom Topic Questions**

Objectives

Students will be able to:

- Explain the purpose of a `void` method
- Write method calls using correct Java syntax

Preparation

- 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

- **Calling Methods** - Video
- **U1L4 Extra Practice** - Handout

Vocabulary

- **Procedural Abstraction** - allows a programmer to use a method by knowing what the method does even if they do not know how the method was written
- `void` **method** - a method that performs an action but does not return a value
- **argument** - a value passed to a method or constructor when the method or constructor is called
- **dot operator** - used to call a method in a class

- **method** - a named set of instructions to perform a task
- **parameter** - a variable in the method or constructor signature that defines the type of value to receive when the method or constructor is called

Teaching Guide

Warm Up (10 minutes)

Giving Instructions

Remarks

> Besides defining classes and creating objects, software engineers write instructions to tell Java how to complete tasks. When we give instructions, we have to be as clear and concise as possible to ensure that Java completes the task the way we intend. We do this in our day-to-day lives as well.

Group: Place students in pairs.

Do This: Have students work with their partner to write instructions that a GPS could use to direct a delivery driver to move from 1st Street and 1st Avenue to the customer located at 5th Street and 5th Avenue.

Teaching Tip

Call on a couple of groups to share their solutions with the class.

Do This: Have students write instructions with their partner that a GPS could use to direct a bus driver to pick up riders from two bus stops, 5th Street and 5th Avenue and 5th Street and 3rd Avenue, and transport them to the library located at 2nd Street and 3rd Avenue.

Activity (30 minutes)

Calling Methods in Java (15 minutes)

Remarks

> In the previous lesson, we learned how to instantiate `Painter` objects. We also looked at a UML diagram for the `Painter` class to learn what attributes a `Painter` has and what a `Painter` can do. But how do we give the `Painter` instructions to perform these behaviors?

Do This: Review the lesson objectives.

Do This: Direct students to Level 1 on Code Studio to predict the program's outcome, then run the program to compare their predictions to the actual outcome.

1 Predict: Painter Class Methods

Discuss: Click through the animated slide to display the prompts.

- *What do you notice about the code in this program?*
- *What do you wonder about the code in this program?*

Discussion Goal: Students identify the methods used in the program and share ideas about what each method does. Students should also notice that each statement executes in the order written in the program. Students may wonder what might happen if the method calls were rearranged or how they might call other methods.

Teaching Tip

To help bring students attention from the level to the discussion, establish a routine to help minimize distractions. For example, you can have students put their laptops to 45 degrees or turn their monitors off. You may also choose to show the Predict levels to the class and share predictions together.

Remarks

> This program uses some of the behaviors that the `Painter` class has. In Java, we represent the things an object can do with methods. We can call these methods to perform a specific task.

Display: Show the video – *Calling Methods*.

Discuss: Use the Hold That Thought strategy to discuss the prompt.

- *Look at these methods in the UML diagram for the* `Painter` *class. What keyword is in front of the method name? Where have you seen this word before?*

Discussion Goal: Students identify the `void` keyword before the method name. Students share ideas about what `void` means and connect other uses of the word from non-programming scenarios, such as emptiness or canceling something.

Teaching Tip

The Hold That Thought strategy allows students to write down their thoughts on paper first. Emphasize that they do not need to write complete sentences – they can write a few words, bullet points, draw a quick sketch, or anything else that will help them gather and recall their thoughts when it is time to share with the class.

Do This: Define `void` *method*.

Remarks

> These void methods can modify the state of a `Painter` object. Let's take a look at the code for the `takePaint()` method. Some of this code may look unfamiliar to you right now, and that's ok! Software engineers often encounter code that is unfamiliar to them, but remember that one of the characteristics of software engineers is that they are learners! We will be learning more about these statements throughout this course.

Do This: Click through the animated slide to discuss the code for the `takePaint()` method.

Teaching Tip

Focus on the code that is highlighted on the slide rather than discussing the unfamiliar components of the code segment. The goal is to give students a glimpse at some of the code that makes up the `Painter` class and to develop code comprehension skills. Emphasize to students that software engineers often encounter code that is unfamiliar to them at first and remind students that one of the characteristics of software engineers is that they are learners. Students learn about the specific purpose and functionality of these code segments in later lessons.

Remarks

> We can use methods that have already been written for us without needing to know how it was written. This is thanks to a concept called procedural abstraction.

Do This: Define *procedural abstraction*.

Do This: Click through the animated slide to explain how to call a method.

Do This: Explain the `paint()` method.

Remarks

> When we call a method or constructor, the program has to first execute the statements in the method or constructor before continuing.

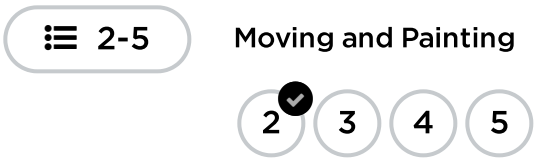
Do This: Click through the animated slide to demonstrate the flow of control when calling a method or constructor.

Using the Painter Class (15 minutes)

Remarks

> Let's practice calling these methods to move around and paint The Neighborhood!

Do This: Direct students to Level 2 on Code Studio to complete Levels 2 through 5. Students debug the program on Level 2, then continue to complete Level 3. Students complete a choice level on Level 4. Students debug the program on Level 5.



Teaching Tip

Choice levels allow students to choose a problem to complete based on the level of complexity or personal interest. Some choice levels are differentiated by complexity, while others are similar levels of complexity but differentiated by topic. Students only need to complete one level but can complete multiple for additional practice if they choose.

Wrap Up (5 minutes)

Finding Errors

Remarks

> Software engineers spend a lot of time debugging errors in their programs, just like those you encountered. Let's use our new skills to help this programmer fix their code.

Do This: Have students identify the cause of the error and suggest what to add or change to correct the error.

Teaching Tip

Prompt students to check the code segment one line at a time to identify potential errors. Ask guiding questions about syntax rules and whether each line follows these rules, such as:

- *Are there matching opening and closing curly braces?*
- *Are there semicolons at the end of each statement?*
- *Are there parentheses when calling constructors and methods?*
- *Are paint colors written with quotation marks?*

Students should identify the missing parentheses on Line 6 and missing quotation marks around `green` on Line 7.

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.

6 **Check for Understanding**

AP Classroom Topic Questions

To assign questions from the AP Classroom Question Bank that align with this lesson, create a custom quiz in AP Classroom by searching the Question Bank for the Essential Knowledge statements listed at the top of this lesson plan. You can find instructions and video demonstrations to do this on **AP Central**.

The following Topic Questions in AP Classroom can be assigned as a formative assessment for this lesson:

- Topic Questions 2.3

Note: *Some Learning Objectives and Essential Knowledge statements in the suggested Topic Questions are covered in later units.*