Lesson 6: PainterPlus

45 minutes

Overview

Why would I use inheritance?

Students consider designing specialized types of classes and use inheritance to create the `PainterPlus` class. Students learn to create a new class and write a constructor that calls the superclass constructor using the `super` keyword. Students realize that while the `PainterPlus` class can perform the same behaviors as the `Painter` class, they can add new behaviors to expand its capabilities.

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
- ▶ **MOD-3** - When mulitple classes contain common attributes and behaviors, programmers create a new class containing the shared attributes and behaviors forming a hierachy. Modifications made at the highest level of the hierarchy apply to the subclasses.

Agenda

Warm Up (10 minutes)
Managing Social Media Accounts

Activity (30 minutes)
Creating a New Class
Creating PainterPlus Objects

Wrap Up (5 minutes)
PainterPlus Behaviors

Objectives

Students will be able to:

- Explain the purpose and functionality of inheritance
- Write a constructor in a subclass that calls the constructor in the superclass
- Write a subclass that extends a superclass

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

- **Inheritance** - Video
- **U1L6 Extra Practice** - Handout

Vocabulary

- `super` **keyword** - a keyword used to refer to the superclass
- **constructor signature** - the first line of the constructor which includes the `public` keyword, the constructor name, and the values to specify when an object is created
- **inheritance** - an object-oriented programming principle where a

Assessment: Check for Understanding
AP Classroom Topic Questions

subclass inherits the attributes and behaviors of a superclass
- **subclass** - a class that extends a superclass and inherits its attributes and behaviors
- **superclass** - a class that can be extended to create subclasses

Teaching Guide

Warm Up (10 minutes)

Managing Social Media Accounts

Remarks

> We have learned a lot about creating classes and objects so far. Software engineers sometimes need to create specific types of objects that share common attributes and behaviors and have additional features. Let's take a look at an example scenario.

Discuss: Click through the animated slide to display the problem and prompts. Use the Retrieve-Pair-Share strategy to discuss the prompts.

- *You are designing a program to keep track of the different social media services that you use. You create a class called* `SocialMedia` *to represent all service media accounts with the attributes* `username` *and* `followers` *as well as a method to* `postMessage`*.*
- *Does this* `SocialMedia` *class represent all the types of social media accounts you use?*
- *What other types of social media services are there? What are the similarities and differences between these types of social media services?*

Discussion Goal: Students identify different types of social media services, such as services that let them post images and messages and services that let them post videos. Students suggest similarities and differences between the types of club members, such as character limits of some services or only posting videos and not messages.

Activity (30 minutes)

Creating a New Class (15 minutes)

Remarks

> In this scenario, we might want to create types of social media services based on their specific capabilities. For example, we might want to create a `VideoSocialMedia` class with additional behaviors for posting and sharing videos. However, a video social media service might still share the same attributes and behaviors as other social media services. To solve this problem, we can use a concept called inheritance.

Do This: Review the lesson objectives.

Display: Show the video – *Inheritance*.

Remarks

> We're going to use inheritance to create a new type of `Painter` called `PainterPlus`. The `PainterPlus` class will extend the `Painter` class and have its own unique behaviors.

Do This: Click through the animated slide to demonstrate that the `PainterPlus` class will inherit the attributes and behaviors from the `Painter` class.

Do This: Direct students to create a UML diagram for the `PainterPlus` class.

Teaching Tip

Remind students of the components of a UML diagram (attributes and behaviors). Emphasize that the `PainterPlus` class will share the same attributes and behaviors as the `Painter` class, so these do not need to be added to `PainterPlus`. Encourage students to brainstorm behaviors they would like to add to `PainterPlus` on their UML diagrams.

Remarks

> You might be wondering why we don't just add what we want to the `Painter` class instead of extending it to create a subclass. In this case, since the `Painter` class is part of The Neighborhood package, we can't modify it. While the `Painter` class lets us navigate and paint The Neighborhood, it has limitations. Extending the `Painter` class to create `PainterPlus` lets us use the behaviors we already have and add new ones.

Creating PainterPlus Objects (20 minutes)

Do This: Click through the animated slide to show how to create a new class in Java Lab.

Remarks

> While a subclass inherits the attributes and behaviors of the superclass, it does not inherit the constructors in a superclass. We will need to write a constructor in the `PainterPlus` class so we can create `PainterPlus` objects.

Do This: Click through the animated slide to review the constructor for the `Painter` class and how it is called to create a `Painter` object.

Remarks

> Let's make our `PainterPlus` class!

Do This: Direct students to Level 1 on Code Studio to complete Levels 1, 2, and 3. Students complete a Check for Understanding on Level 1, then continue to Level 2 to write the `PainterPlus` class by creating a new file and writing the class header and constructor. Students create a `PainterPlus` object on Level 3.

1-3 **Creating PainterPlus**

1 2 3

Wrap Up (5 minutes)

PainterPlus Behaviors

Remarks

> Right now, our `PainterPlus` objects can only do the same things a `Painter` object can do. In the next lesson, we will add a new method to the `PainterPlus` class so that our `PainterPlus` objects can do things that a `Painter` object can't do.

Discuss: *What are some things you want a* `PainterPlus` *object to do that a* `Painter` *object can't do?*

Discussion Goal: Students share ideas of methods they could add to the `PainterPlus` class, such as a method to turn right, navigate faster, or take all of the paint from a paint bucket.

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

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 9.1
- Topic Questions 9.4

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

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**.