Designing Classes

The String class provides methods for working with text. The Random class provides methods for generating random numbers. In this activity, you'll learn how to make your own classes that represent everyday objects.

Manager: **Self**

Recorder:

Presenter:

Reflector:

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the purpose of constructor, accessor, and mutator methods.
- Implement the equals and toString methods for a given class design.
- Design a new class (UML diagram) based on a general description.

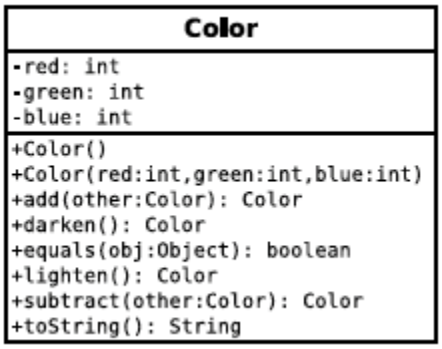
Process Skill Goals

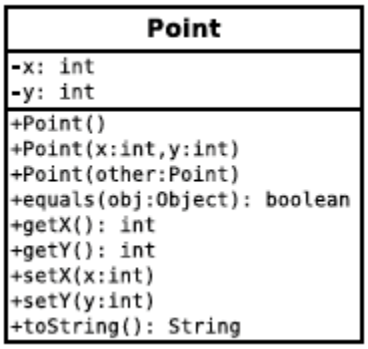
During the activity, students should make progress toward:

- Identifying attributes and data types that model a real-world object. (Problem Solving)

Model 1 Common Methods

Classes are often used to represent abstract data types, such as Color or Point:





As shown in the UML diagrams, classes generally include the following kinds of methods (in addition to others):

- **constructor** methods that initialize new objects
- **accessor** methods (getters) that return attributes
- **mutator** methods (setters) that modify attributes

Questions (15 min)

1. Identify the constructors for the Color class. What is the difference between them?

Color() and Color(red, green, blue)
One creates a Color instance without r,g,b parameters set; the other creates a Color instance with those parameters set

2. What kind of constructor does the Point class have that the Color class does not?

Point(other)

3. Identify an accessor method in the Point class.

a) What is the name of the method? **getX()**

b) Which instance variable does it get? **this.x**

c) What arguments does the method take? **none**

d) What does the method return? **The value of x**

4. Identify a mutator method in the Point class.

   a) What is the name of the method? **setX()**

   b) Which instance variable does it set? **this.x**

   c) What arguments does the method take? **A new value of x**

   d) What does the method return? **Nothing (void)**

5. How would you define accessor methods for each attribute of the Color class? Write your answer using UML syntax.

+ getRed(): int
+ getBlue(): int
+ getGreen(): int

6. How would you define mutator methods for each attribute of the Color class? Write your answer using UML syntax.

+setRed(newRed: int)
+setBlue(newBlue: int)
+setGreen(newGreen: int)

7. The Color class does not provide any accessors or mutators. Instead, it provides methods that return new Color objects. Why do you think the class was designed this way? 🔑

The lack of setters and getters implies that the Color class is immutable, which means individual Color objects can be reused (just like String).

Model 2 Object Methods

In addition to providing constructors, getters, and setters, classes often provide equals and toString methods. These methods make it easier to work with objects of the class.

As a team, review the provided *Color.java* and *Point.java* files. Run each program to see how it works. Then answer the following questions using the source code (don't just guess).

Questions (15 min)

8. Based on the output of *Color.java*, what is the value of each expression below?

```
Color black = new Color();
Color other = new Color(0,0,0);
Color gold = new Color(255, 215, 0;
```

a) black == other → **false**
b) black == gold→ **false**
c) black.toString()→ **#000000**
d) black.equals(other) → **true**
e) black.equals(gold) → **false**
f) gold.toString()→ **#ffd700**

9. What is the purpose of the toString method?

It serializes a Color object into a string (in the form of a hex code).

10. Based on the output of *Point.java*, what is the value of each expression below?

```
Point p1 = new Point();
Point p2 = new Point(0, 0);
Point p3 = new Point(3, 3);
```

a) p1 == p2 → **false**
b) p1.toString() → **(0, 0)**
c) p3.toString() → **(3, 3)**
d) p1.equals(p2) → **true**
e) p1.equals("(0,0)") → **false**
f) p3.equals("(3,3)") → **false**

11. What is the purpose of the equals method? 🔑

The equals method checks whether two Points have the same x and y attributes.

Examine *Point.java* again. What is the purpose of the if-statement in the equals method?
The if statement validates that the Object passed into equals() is a Point.

12. How could you modify the equals method to cause both #10e and #10f to return true?
Change return false; to return this.toString().equals(obj);

Model 3 Credit Card

Classes often represent objects in the real world. In this section, you will design a new class that represents a CreditCard like the one below:

Questions (15 min)

13. Identify four or more **attributes** that would be necessary for the CreditCard class. For each attribute, indicate what data type would be most appropriate.

long cardNumber, int cvv, String name, int expirationYear, int expirationMonth

14. Using UML syntax, define two or more **constructors** for the CreditCard class.

+CreditCard() // randomly generates values of attributes
+CreditCard(cardNumber: long, cvv: int, name: String, expirationYear: int, expirationMonth: int)

15. Define two or more **accessor** methods for the CreditCard class. Include arguments and return values, using the same format as a UML diagram.

+getCardNumber(): long
+getCVV(): int

16. Define two or more **mutator** methods for the CreditCard class. Include arguments and return values, using the same format as a UML diagram.

+setCardNumber(newCardNumber: long)
+setCVV(newCVV: int)

17. Describe how you would implement the equals method of the CreditCard class. 🔑

The equals method would check that the cardNumber value of each CreditCard being compared is equal (with the real-life assumption that cardNumbers are not duplicated).

18. Describe how you would implement the toString method of the CreditCard class.

The toString method would print the card number, CVV, expiration month, expiration year, and cardholder name - all concatenated into one String with commas separating each value.

19. When constructing (or updating) a CreditCard object, which arguments would you need to validate? What are the valid ranges of values for each attribute?

cardNumber: Should be exactly 16 digits and we should use the Luhn algorithm to validate that the check digit matches the expected value.

CVV: Should be exactly 3 to 4 digits (000 to 9999)

expirationYear: Should be greater than this year (24) and two digits

expirationMonth: Should be (01-12) and two digits