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: 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.

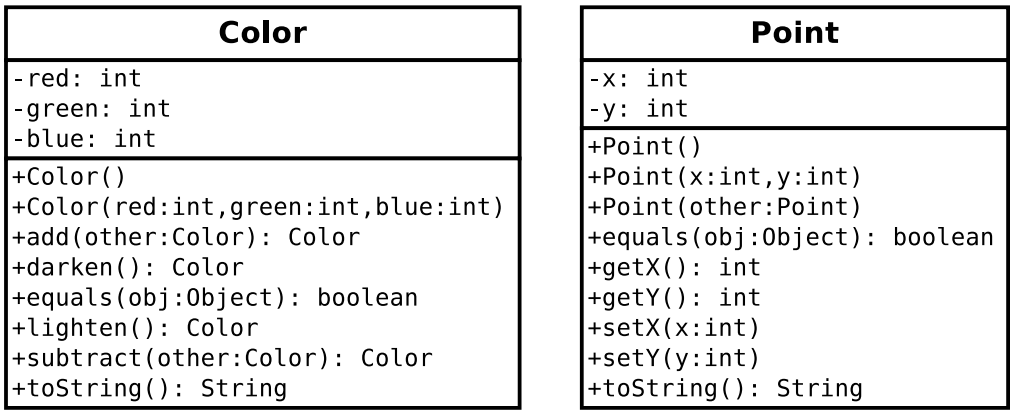
Process Skill Goals

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) Start time:

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

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

3. Identify an accessor method in the `Point` class.

a) What is the name of the method?

b) Which instance variable does it get?

c) What arguments does the method take?

d) What does the method return?

4. Identify a mutator method in the `Point` class.

a) What is the name of the method?

b) Which instance variable does it set?

c) What arguments does the method take?

d) What does the method return?

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

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

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?

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) Start time:

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`

b) `black == gold`

c) `black.toString()`

d) `black.equals(other)`

e) `black.equals(gold)`

f) `gold.toString()`

9. What is the purpose of the `toString` method?

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`

b) `p1.toString()`

c) `p3.toString()`

d) `p1.equals(p2)`

e) `p1.equals("(0, 0)")`

f) `p3.equals("(3, 3)")`

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

12. Examine *Point.java* again. What is the purpose of the `if`-statement in the `equals` method?

13. How could you modify the `equals` method to cause both #10e and #10f to return `true`?

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) Start time:

14. Identify two or more attributes that would be necessary for the `CreditCard` class. For each attribute, indicate what data type would be most appropriate.

15. Using UML syntax, define two or more constructors for the `CreditCard` class.

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

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

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

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

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