Classes and UML

When you define a class in Java, you are designing a new type of object. Each object has its own copy of the variables and methods in the class.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Define the terms: attribute, method, constructor, scope.
- Implement non-static methods based on a UML diagram.
- Distinguish static, instance, parameter, and local variables.

Process Skill Goals

During the activity, students should make progress toward:

- Writing method signatures exactly as shown in a UML diagram. (Information Processing)

Facilitation Notes

Model 1 is primarily about introducing object-oriented vocabulary. As you walk around, keep an eye on #4. Some students may incorrectly write `int lucky`, and if that happens, have them re-read the text in the model. For reporting out, quickly have presenters of neighboring teams compare their answers and ask questions as needed.

Model 2 asks students to implement a `Circle` class, one method at a time. Make sure they write the complete method definition, not just the method body. After students complete #15, show *Circle.java* on the projector and step through the code beginning from `main`. It may be helpful to use Java Visualizer to illustrate how `Circle` objects are created.

Model 3 presents a slightly different circle implementation named `SwapCircle`. Have students work through the first questions as quickly as possible so they can spend time on #20. You'll need about ten minutes at the end of class to walk students through the solution, again using Java Visualizer or a similar tool.

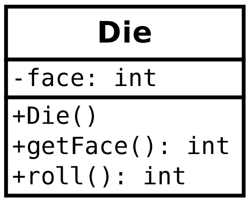
Key questions: #4, #15, #20

Source files: *Die.java*, *SwapCircle.java*, *SwapDriver.java*

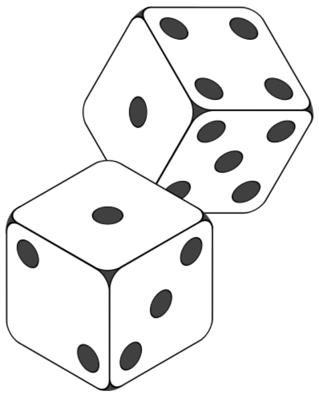
Model 1 The Die Class

The following class represents an individual "die" in a game of dice. The diagram on the right is a graphical summary of the ***attributes*** (variables) and ***methods*** of the class.

```java
/**
 * Simulates a die object.
 */
public class Die {

    private int face;

    /**
     * Constructs a die with face value 1.
     */
    public Die() {
        this.face = 1;
    }

    /**
     * @return current face value of the die
     */
    public int getFace() {
        return this.face;
    }

    /**
     * Simulates rolling the die.
     *
     * @return new face value of the die
     */
    public int roll() {
        this.face = (int) (Math.random() * 6) + 1;
        return this.face;
    }

}
```



```
Die
-face: int
+Die()
+getFace(): int
+roll(): int
```



Questions (10 min) Start time:

1. Consider the `Die` class:

a) What are the attributes? face

b) What are the methods? Die, getFace, roll

2. In the class diagram (on the upper right):

a) What do the `+` and `-` symbols represent? + means `public` - means `private`

b) What does the `:` represent? the data type of the attribute/method

3. Open the provided *Die.java* and run the program several times. Then answer the following questions about the `main` method:

a) What is the data type of `d1` and `d2`? Die

b) What are the initial values of the dice? d1 = 1, d2 = 1

c) What method changed the dice values? roll

4. Write a statement that declares and initializes a `Die` variable named `lucky`.

```
Die lucky = new Die();
```

5. When you create an object, Java invokes a ***constructor***. This method has no return type and has the same name as the class itself. What does the `Die()` constructor do?

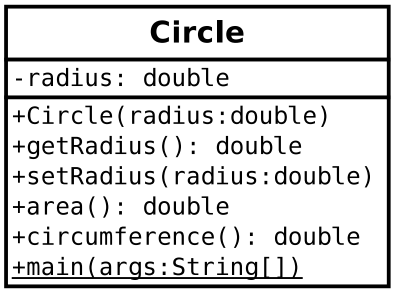
It initializes the `face` attribute to 1. (Without this constructor, the default value would be 0, which is invalid for dice.)

6. Notice how the `roll` method refers to `this.face`, yet that variable is not declared in the method. What does the `roll` method change, in terms of the `Die` object?

It updates the value of the `face` attribute.

Model 2 The Circle Class

Unified Modeling Language (UML) provides a way of graphically illustrating a class's design, independent of the programming language.



```
Circle
-radius: double
+Circle(radius:double)
+getRadius(): double
+setRadius(radius:double)
+area(): double
+circumference(): double
+main(args:String[])
```

Questions (15 min) Start time:

7. Consider the `Circle` class:

a) How many attributes does it have? 1

b) How many methods does it have? 6

8. Based on Model 1 and Model 2, what is typically `public` and what is typically `private`?

Methods are typically public, and attributes are typically private.

The following questions will have you implement the `Circle` class exactly as shown in the UML diagram above. Do not worry about writing Javadoc comments for this activity.

9. Write the code that declares the `radius` attribute. An outline of *Circle.java* is provided below for context.

```
public class Circle {
    private double radius;

    // constructor goes here

    // other methods go here
}
```

10. Write the code for the `Circle` constructor. Notice that, in contrast to Model 1, the `Circle` constructor has a parameter. Assign the parameter `radius` to the attribute `this.radius`.

```
public Circle(double radius) {
    this.radius = radius;
}
```

11. Write the code for `getRadius`. (Refer to Model 1 for an example.)

```
public double getRadius() {
    return this.radius;
}
```

12. Write the code for `setRadius`. Like the constructor, it should assign the parameter to the corresponding attribute.

```
public void setRadius(double radius) {
    this.radius = radius;
}
```

13. Write the code for `area`. The area of a circle is πr^2.

```
public double area() {
    return Math.PI * radius * radius;
}
```

14. Write the code for `circumference`. The circumference of a circle is $2\pi r$.

```
public double circumference() {
    return 2.0 * Math.PI * radius;
}
```

15. Write a `main` method that creates a `Circle` object with a radius of 2.0 and displays its area and circumference (using `println`).

```
public static void main(String[] args) {
    Circle circle = new Circle(2);
    System.out.println(circle.area());
    System.out.println(circle.circumference());
}
```

Model 3 Variable Scope

As a team, review and discuss the provided *SwapCircle.java* and *SwapDriver.java* source files. Then identify the ***scope*** of each variable (i.e., where it can base used) based on the table below.

	Where declared?	**Where used?**	**Example**
static variables ("class variables")	declared outside of all methods (typically at the start of the class)	anywhere in the class (or in other classes if also `public`)	`circleCount` in the `SwapCircle` class
instance variables ("attributes")	declared outside of all methods (typically after any static variables)	any non-static method (or in static methods when another object has been created)	`radius` in the `SwapCircle` class
parameters	declared inside the ()'s of a method signature	anywhere within the method where they are declared	`radius` in the `SwapCircle` constructor
local variables	declared inside a method (or inside another block of code, like a `for` loop)	anywhere within the method or code block after they are declared	`temp` in the `swapInts` method

Questions (20 min)

Start time:

16. Identify one static variable from the `SwapCircle` class.

a) What is the name and purpose of the variable?

`circleCount` – tracks the number of `SwapCircle` objects that have been created

b) What is the scope of the variable?

`private static` – it can be used anywhere within the `SwapCircle` class only

c) What is one example of somewhere it cannot be used?

`SwapDriver.main`

17. Identify one instance variable from the `SwapCircle` class.

a) What is the name and purpose of the variable?

`radius` – stores the radius of `this` SwapCircle

b) What is the scope of the variable?

`private` and non-`static` – it can only be used in `SwapCircle` in non-static contexts

c) What is one example of somewhere it cannot be used?

`SwapDriver.main`

18. Identify one parameter from the `SwapCircle` class.

a) What is the name and purpose of the variable?

Possible answers include: `radius` (in the constructor), `x` and `y`, `c1` and `c2`

b) What is the scope of the variable?

The variable exists throughout the entire method (but not other methods).

c) Where can the variable not be used?

It can't be used in other methods, e.g., you can't refer to `x` in `swapCircles`.

19. Identify one local variable from the `SwapCircle` class.

a) What is the name and purpose of the variable?

Possible answers include: `temp` and `r` (both used for swapping values)

b) What is the scope of the variable?

The variable exists from when it's declared until the end of the method.

c) Where can the variable not be used?

It cannot be used in other methods and/or before it has been declared.

20. Run the `SwapDriver` program and summarize what you learn based on the output.

Answers will vary. The primitive integers were not swapped, but the attributes of the `Circle` objects were. The `static circleCount` kept track of how many objects were created.

21. Notice that `getRadius` returns `this.radius` (from `this` object). In contrast, `getCircleCount` does not use the keyword `this`. Why not?

getCircleCount is a static method, so there is no object. If you try to use `this`, you will get a compiler error that says, "non-static variable this cannot be referenced from a static context".

22. Identify an example of where an instance variable is used within a static method.

a) In which method does this occur?

`radius` is used in the `swapCircles` method

b) Why is the method able to access these instance variables, even though they are private?

`swapCircles` belongs to the `SwapCircle` class

c) Explain how this method is not a violation of the rule that instance variables cannot be accessed inside a static method.

You can't use `this`.radius in a `static` method, but it's okay to use `c1.radius`