Introduction to Objects and Debugging

Today, we look at how to create and apply custom object types in Java.

Content Learning Targets

After completing this activity, you should be able to say:

- I can define objects and related terms, such as field, attribute, method, constructor.
- I can identify the fields and methods of a Java object.
- I can create new object types in Java.
- Given a Java class, I can construct the corresponding UML class diagram.
- Given a UML class diagram, I can write the corresponding Java code.
- I can use debugging tools.

Process Skill Goals

During the activity, you should make progress toward:

- Verifying program behavior by tracing code with a debugger. (Critical Thinking)

Facilitation Notes

In Model 1, students analyze a given Java class representing a six-sided die.

Model 2 introduces UML class diagrams and how to translate them into code.

Model 3 introduces the why/what/how of the debugger and ends with debugging practice.

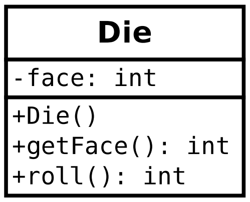
Key questions: #4, #15, #16

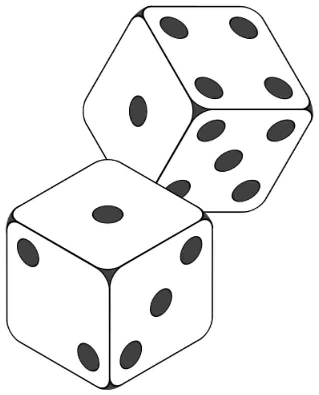
Source files: *Die.java*, *debugger/DebugMe.java*, *debugger/DebugMeTest.java*, *debugger/WhackABug.java*

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;
    }

}
```





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?

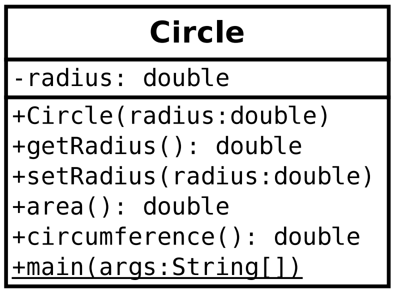
It initializes the `face` attribute to 1. (Without this constructor, the default value would be 0, which is invalid for dice.)

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 Debugging

Questions (25 min) Start time:

16. Discuss (2 min): If a program is not working the way you expect (e.g., it is not passing all its tests), what are some options for figuring out what's wrong?

Expected answers: change some statements and see what happens, print statements to see intermediate variable values, read Java error messages (if present), look back through code and think through its logic

A good Integrated Development Environment (IDE) provides a ***debugger***, a suite of tools for finding and fixing errors in code. After the debugger demonstration, answer these questions.

17. What is the term for a place in which we tell the debugger to pause program execution?

breakpoint

18. After the debugger pauses, what primary commands do you have available?

Resume (to keep going as normal), Terminate (to stop entirely), Step Into (to go to the first line of the invoked method, if any), and Step Over (to go to the next line in the current method)

19. If you get some type of Java Exception but aren't sure where or why, what can you do?

Set an ***exception breakpoint*** that will pause execution whenever the program encounters an exception of the specified type.

20. Open ***DebugMe.java*** and ***DebugMeTest.java***. Practice using the debugger to find and fix the bug in `uppercaseIfExclamation`.

Java Strings are immutable, so calling `toUpperCase` does not change `sentence`. We need to return (or store and return) the modified String.

21. Practice using the debugger to find and fix the bugs in `kPermutations` and `isArrayDoubled`.

In `kPermutations` we encounter an `int` overflow. In `isArrayDoubled` we realize the `equals` method does not compare arrays' values.

22. Continue debugging practice with ***WhackABug.java***. Note the bugs and lessons learned.

See the ***WhackABug.java*** copy in the solutions folder for explanations of the bugs.