# Introduction to Objects and Debugging

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

Manager: Recorder:

Presenter: Reflector:

## 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, cosntructor.
- 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)

# Meta Activity: What Employers Want

These data are from the *Job Outlook 2025* survey by the National Association of Colleges and Employers (NACE). A total of 237 organizations responded to the survey.

**Attributes Employers Seek on a Candidate's Resume**

| Attribute | % of respondents |
|---|---|
| Ability to work in a team | 81.0% |
| Analytical/quantitative skills | 67.0% |
| Communication skills (verbal) | 69.3% |
| Communication skills (written) | 77.1% |
| Detail-oriented | 65.9% |
| Initiative | 73.7% |
| Leadership | 52.5% |
| Problem-solving skills | 88.3% |
| Strong work ethic | 73.2% |
| Technical skills | 73.2% |

Source: https://www.naceweb.org/talent-acquisition/candidate-selection/

## Questions (10 min) Start time:

**1**. What are the top three attributes that employers look for on a resume?

#1: #2: #3:

**2**. How is communication (written and verbal) related to problem solving and teamwork?

**3**. How does the team-based learning approach in this class help you develop these skills?

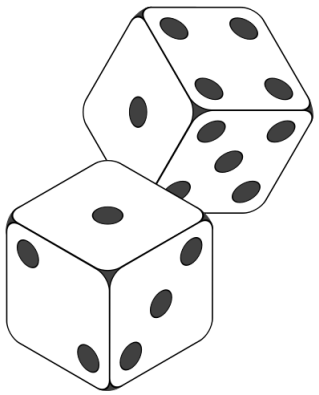
**4**. Do you find the data surprising? If so, in what way(s)? Why do you think employers have these priorities?

# 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.

```
/**
 * 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()<br>+getFace(): int<br>+roll(): int |



## Questions (10 min) Start time:

**5**. Consider the `Die` class:

a) What are the attributes?

b) What are the methods?

**6**. In the class diagram (on the upper right):

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

b) What does the `:` represent?

**7**. 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`?

b) What are the initial values of the dice?

c) What method changed the dice values?

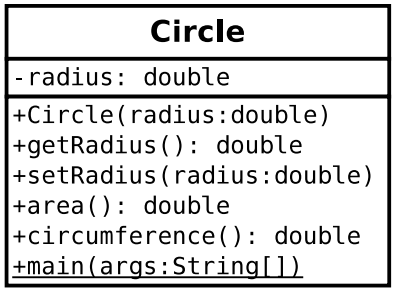
**8**. Write a statement that declares and initializes a `Die` variable named `lucky`.

**9**. 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?

**10**. 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?

# Model 2 The Circle Class

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




## Questions (15 min) Start time:

**11**. Consider the `Circle` class:

a) How many attributes does it have?

b) How many methods does it have?

**12**. Based on Model 1 and Model 2, what is typically `public` and what is 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.*

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

```
public class Circle {


    // constructor goes here

    // other methods go here
}
```

**14**. 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`.

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

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

**17**. Write the code for `area`. The area of a circle is $\pi r^2$.

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

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

# Model 3 Debugging

## Questions (30 min. (+15 min. demo)) Start time:

**20**. 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?

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.

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

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

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

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

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

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