Introduction to Java Classes and Objects

COP2250: Java Programming

Professor Pyatt

Week of January 13, 2026

Today's Objectives

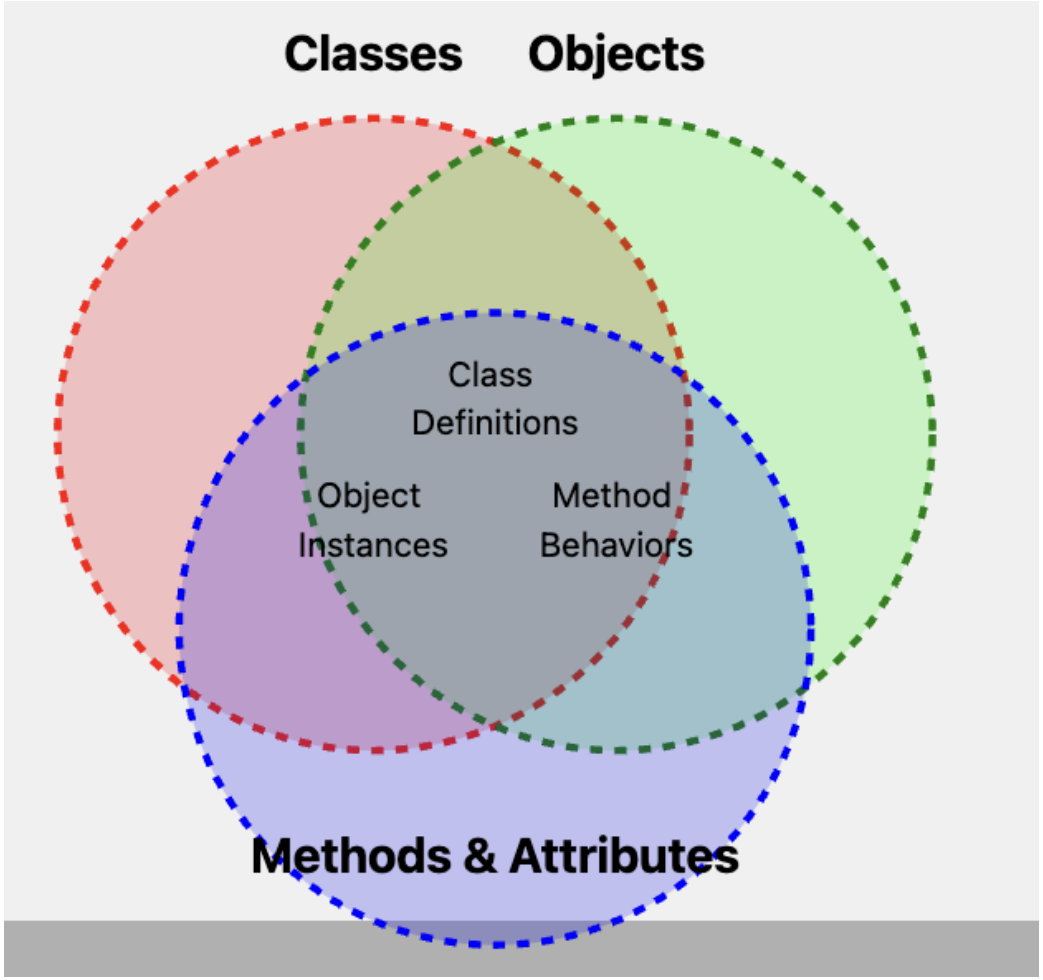
- Understand classes as blueprints for objects
- Create objects (instances) from classes
- Define and call methods
- Build a complete Vending Machine application
- Push your project to GitHub

Real-World Analogy: Vending Machine

Scenario: Create a simple Java program using Objects, Methods, and Classes to represent a Vending Machine.

- A Vending Machine is like a Java **Class**
- Selecting a snack or drink is like calling a **Method**
- The class knows how to dispense items
- **Class**: A template for objects
- **Object**: An instance of a class
- **Method**: A block of code that performs a specific task

Classes, Objects, Methods



Step 1: Define a Class

```
// VendingMachine.java
class VendingMachine {

    // Method: performs an
        action
    void dispenseItem(String
        item) {
        System.out.println(
            "Dispensing " + item
                + "...");
    }
}
```

Key points:

- `class VendingMachine` declares the blueprint
- `dispenseItem()` is a method
- `void` means no return value
- `String item` is the parameter
- Methods are reusable: define once, use many times

The Entry Point: main()

The entry point for every Java application is the `main` method:

```
public static void main(String[] args) {
    // Your code starts here
}
```

Breaking it down:

- **public**: Accessible from anywhere
- **static**: Belongs to the class, not an instance
- **void**: Returns nothing
- **main**: The name JVM looks for
- **String[] args**: Command-line arguments

Step 2: Create an Object and Call Methods

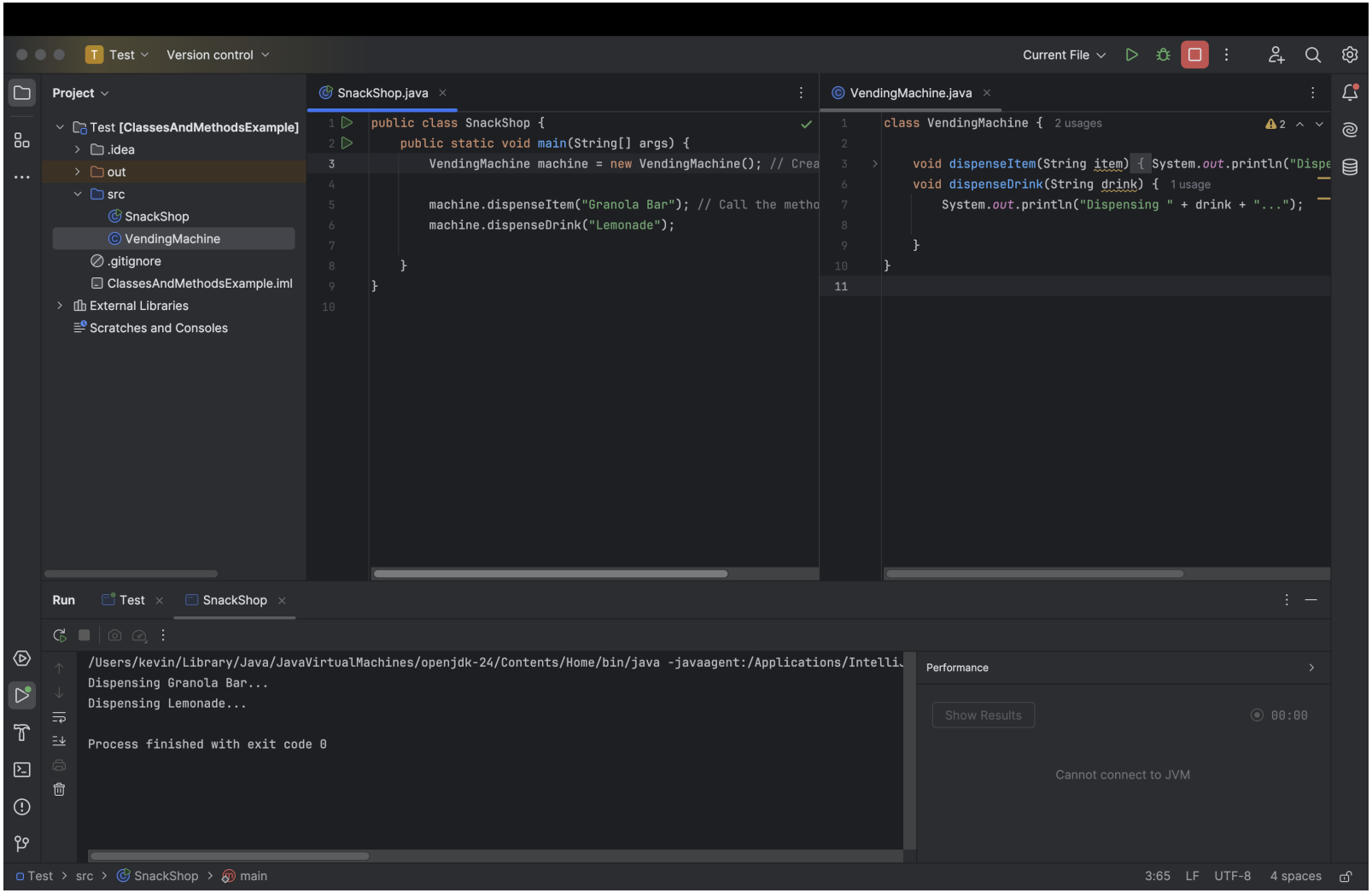
```
// SnackShop.java
public class SnackShop {
  public static void
      main(String[] args) {
    // Create an object
    VendingMachine machine =
        new VendingMachine();

    // Call the method
    machine.dispenseItem("Chips");
  }
}
```

Key points:

- JVM looks for this exact method signature
- `new VendingMachine()` creates an object
- `machine` holds the object reference
- `machine.dispenseItem("Chips")` calls the method

Dev Environment and Compiling



Code Challenge: Add More Methods

Challenge: Modify the VendingMachine class to add a method for drinks.

```
// Add to VendingMachine class
void dispenseDrink(String
    drink) {
    System.out.println(
        "Pouring " + drink +
            "...");
}
```

Then call it:

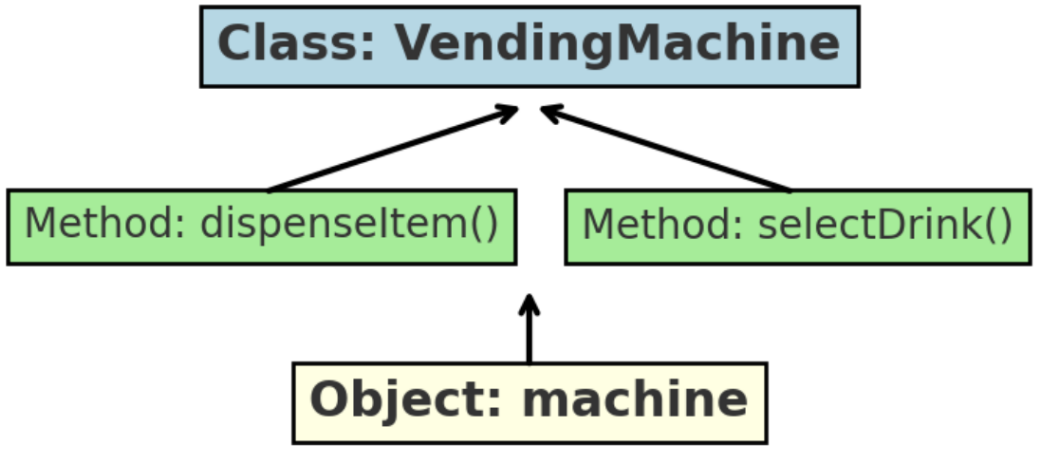
```
// Add to main method
machine.dispenseDrink("Lemonade");
```

Each method handles a different action the machine can perform.

Why This Matters

Why use methods and classes?

- Methods make code reusable and organized
- Classes create blueprints for objects
- These concepts are essential for Object-Oriented Programming (OOP)



Summary: Putting It All Together

Feature	Class	Object	Method
Definition	A blueprint for creating objects	An instance of a class	A function inside a class
Purpose	Defines the structure and behavior	Represents a real-world entity	Defines behavior or actions
Example	`VendingMachine`	`machine = new VendingMachine();`	`dispenseItem()`
Contains	Methods and variables	Instance variables and methods	Code that executes a task
Usage	Used to create objects	Interacts with methods	Called using an object

Lab: Build Your Vending Machine

Create a complete vending machine application:

1. Create `VendingMachine.java` with multiple methods
2. Create `Snack.java` class with name and price
3. Create `SnackShop.java` with main method
4. Instantiate at least 3 different snacks
5. Call methods to dispense items and show prices
6. **Stretch**: Add a `Drink` class

Deliverable: Push completed project to GitHub and share link.

Next Steps

- Complete the lab project and push to GitHub
- Experiment with adding more methods and classes
- Review: classes are blueprints, objects are instances
- **Next week**: Constructors, instance variables, and encapsulation