

Object-Oriented Programming Workshop - Student Handbook

Part One: Introduction to OOP Concepts and Applications

What is Object-Oriented Programming?

Object-Oriented Programming (OOP) represents a fundamental approach to software design where "objects" serve as the building blocks of your code. These objects bundle data with the functions that operate on them. At its heart, OOP allows developers to model real-world entities as software components.

Core Concepts

1. **Classes** - Blueprints that define the structure and behavior of objects
2. **Objects** - Instances of classes with specific properties and behaviors
3. **Encapsulation** - The practice of bundling data with methods and controlling access
4. **Inheritance** - The ability for a class to adopt properties and methods from another class
5. **Polymorphism** - The capacity for different classes to respond uniquely to the same method call

Real-World Applications of OOP

Graphical User Interfaces

Modern applications rely on OOP to create interactive elements like buttons, windows, and menus. Each UI component exists as an object with properties (size, color, position) and behaviors (responding to clicks or hover events).

Game Development

OOP forms the backbone of game design, where characters, enemies, items, and environmental elements exist as objects with their own characteristics and behaviors. This approach naturally maps to the entity-based nature of games.

IoT and Device Control

Smart device ecosystems use OOP to model physical devices in software, where each device class encapsulates the state and control methods specific to that device type.

Simulation and Modeling

From basic physics simulations to complex ecosystem models, OOP excels at representing interacting entities. Each component in the simulation becomes an object that can interact with others according to defined rules.


Database Applications

Modern database interfaces often use Object-Relational Mapping (ORM) techniques, representing database tables and records as objects to streamline data handling and provide a more intuitive programming interface.

Part Two: Getting Started with the p5.js Ball Demo

Let's dive into our interactive physics simulation demo built with p5.js. This project will bring OOP concepts to life through a dynamic ball simulation.

Setting Up the Demo

1. Open your browser and navigate to <https://github.com/quenton1225/Balls>
2. Locate and click on the **balls.txt** file
3. Click the "**Copy raw file**" button to copy the entire codebase
4. In a new tab, open the p5.js web editor: <https://editor.p5js.org/>
5. Replace the default code by pasting in our simulation code
6. Click the  play button in the top-left corner to launch the simulation

Understanding the Base Functionality

Once running, you'll see a canvas populated with colorful, animated balls. This simulation demonstrates several key OOP principles through:

- **The Ball Class:** A parent class establishing basic properties and behaviors
- **The BouncingBall Class:** A specialized ball that rebounds from screen boundaries
- **The GravityBall Class:** A physics-based ball affected by gravity and featuring elastic collisions

Interacting with the Demo

- **Left Click:** Spawns a new ball at the clicked location
- **Key 1:** Switches to creating bouncing balls
- **Key 2:** Switches to creating gravity balls
- **Spacebar:** Applies an upward force to all gravity balls
- **R Key:** Resets the simulation with fresh balls

Take a moment to experiment with these controls and observe how different ball types behave distinctly. These differences showcase inheritance and polymorphism in action.

The OOP Architecture

Ball (Base Class)

- Properties:
 - x, y - Position coordinates
 - r - Radius
 - color - RGB array [255,0,0] (red)
 - xSpeed, ySpeed - Velocity components
- Methods:
 - display() - Renders the ball
 - move() - Updates position
 - checkBoundaries() - Handles screen edge interactions

BouncingBall (Child Class)

- Properties:
 - Inherits all Ball properties
 - Customizes xSpeed, ySpeed via constructor
 - Sets color to [0,0,255] (blue)
- Methods:
 - Inherits all Ball methods without modification

GravityBall (Child Class)

- Properties:
 - Inherits Ball properties
 - Implements velocity as a vector
 - Adds acceleration and damping properties
 - Sets color to [255,150,0] (orange)
- Methods:
 - Overrides move() with physics-based movement
 - Implements checkBoundaryCollision()
 - Customizes display() with velocity-based effects
 - Adds applyForce() for external forces

Part Three: Creating Custom Balls with AI Assistance

Now that you understand the existing simulation, it's time to extend it by creating your own custom ball types. This hands-on exercise will reinforce your understanding of inheritance and polymorphism.

Project Overview

The existing codebase includes:

- **Ball**: The foundational class with basic movement
- **BouncingBall**: Adds boundary reflection behavior
- **GravityBall**: Implements realistic physics with gravity

Choose Your Challenge

Select one of these custom ball types to implement:

Pulsing Ball

- Create a ball that rhythmically changes size
- Consider what properties would control the pulsing (minimum/maximum size, pulse rate)
- Think about how to modify the `display()` method to visualize the size variation

Rainbow Ball

- Design a ball that cycles through colors as it moves
- Considering the existing ball colors, how would you convert HSB to RGB mode while maintaining smooth color transitions?
- Implement controls to adjust the color cycling speed

Trail Ball

- Develop a ball that leaves a fading trail as it moves
- Design a system to store and display previous positions
- Create a fade effect for the trail elements

Portal Ball

- Build a ball that teleports across the screen when reaching edges
- Modify the boundary checking logic to implement teleportation
- Add visual effects to enhance the teleportation experience

Advanced Challenge: Collision System

- Implement ball-to-ball collision detection and response
- Create an efficient method to detect collisions between balls
- Calculate physically accurate post-collision velocities based on conservation principles

Collaborating with AI: A Three-Stage Approach

Lingnan University provides students with access to Lingnan GPT at <https://chatgpt.ln.edu.hk/> (student login required). Here's an effective workflow for using AI to help develop your custom ball type:

Stage 1: Planning & Analysis

Start by describing your concept to the AI assistant:

I'm developing a physics simulation in p5.js using object-oriented principles. I'd like to create a "[Ball Type Name]" that [describe desired behavior]. I've attached the current code. This new ball should inherit from the Ball class and implement [specific features]. Could you help me identify what properties and methods I'll need? Just list the names and purposes without implementation details.

Remember to upload the balls.txt file as an attachment

Stage 2: Design Development

Once you have an initial outline, request more detailed design guidance:

Thanks for the analysis. I'd like to explore the implementation details:

- 1) What parameters should my constructor accept, and how should it call the parent constructor?*
- 2) For [property name], what would be appropriate initial values and ranges?*
- 3) How should the [method name] function work conceptually?*

Please explain the implementation approach for each component, but hold off on providing complete code.

Stage 3: Implementation

When you're ready for implementation:

Your design approach makes sense. Could you now provide a complete implementation including:

- 1) The full class definition for my new ball type*
- 2) Any global variable additions needed*
- 3) Updates to the createBall() function*
- 4) Any keyPressed() modifications for controlling my ball type*

Please highlight how your implementation demonstrates OOP principles like inheritance and polymorphism. Provide the complete, integration-ready code that I can paste directly into the editor.

Troubleshooting Prompts

If you encounter issues/bugs:

*When I implemented the code, I noticed **[specific issue]**, which doesn't align with the **[intended behavior]**. Could you identify the problem and provide a corrected version of the complete code?*

Remember, the LLM will answer you precisely only if you describe detailly.

If the AI doesn't provide complete code:

Your explanation is helpful, but I need the full implementation to test. Could you provide the entire updated code so I can copy it directly into the editor?

To better understand the implementation:

The implementation works, but I'd like to understand the mechanics better. Could you explain in plain language how this ball achieves its special effects and how it interacts with the existing system?

Remember: While AI can accelerate your development process, take time to understand how each piece of code works. The goal isn't just to create a working demo but to deepen your understanding of object-oriented programming principles.

Resources for Further Learning

p5.js Documentation

- [Official p5.js Reference](#) - Comprehensive documentation of all p5.js functions
- [p5.js Examples Gallery](#) - Inspiration and code samples

OOP Learning Materials

- [MDN's JavaScript Classes Guide](#) - Detailed explanation of class implementation in JavaScript
- [JavaScript.info OOP Section](#) - Clear tutorials on OOP concepts

Programming is as much about exploration as it is about implementation. Don't be afraid to experiment, make mistakes, and refine your approach. While AI can be a valuable partner in your coding journey, the understanding you develop by working through challenges is the true reward. Enjoy the creative process of bringing your ideas to life through code!

Happy coding!