Chapter 7 - Object-Oriented Design

- Step 1 → Handle ambiguity
- Step 2 → Define the CORE objects
- Step 3 → Analyze relationships
- Step 4 → Investigate actions

// **Design Patterns** → Singleton Class

```
public class Restaurant {
	private static Restaurant _instance = null;
	protected Restaurant() {}
	public static Restaurant getInstance() {
		if (_instance == null) {
			_instance = new Restaurant();
		}
		return _instance;
	}
}
```

// **Design Patterns** → Factory Method: offers an interface for creating an instance of a class; its subclasses decide which class to instantiate;

```
public class CardGame {
	public static CardGame createCardGame(GameType type) {
		if (type == GameType.Poker) {
			return new PokerGame();
		} else if (type == GameType.BlackJack) {
			return new BlackJackGame();
		}
		return null;
	}
}
```

// **Deck of Cards** → Design the data structures to hold a generic deck of cards

// **Call Center** → Desing a Call Center system

// **Jukebox** → Design a musical jukebox

// **Parking Lot** → Design a parking lot

// **Online Book Reader** → Design the data structures for an online book reader system

// **Jigsaw** → Implement a N x N jigsaw puzzle

- the puzzle is grid like with rows and columns
- each piece is located in a single row and column and has four edges
- each edge is one of three types: inner, outer and flat
- absolute position versus relative position
- group the pieces into corner pieces, border pieces and inside pieces

// **Chat Server**

- How do we know if someone is online? → periodically ping the client?
- How do we prevent Denial of Service attacks?

// **Othello** → Implement the Object-Oriented design of Othello

- Should we implement Game as a singleton class? → this means it will be instantiated only once
- We implement a Piece class that has a Color variable that we can easily flip
- Player.getScore() → will call the Game object to retrieve this value

// **Circular Array**

```
public class CircularArray<T> {
	private T[] items;
	prinvate int head;

	public CircularArray(int size) {
		items = (T[]) new Object[size];
	}

	private int convert(int index) {
		if (index < 0) {
			index += items.length;
		}
		return (head + index) % items.length;
	}

	public void rotate(int shiftRight) {
		head = convert(shiftRight);
	}
```

```
	public T get(int i) {
		if (i < 0 || i >= items.length) {
			throw new IndexOutOfBoundsException("Out of bounds");
		}
		return items[convert(i)];
	}

	public void set(int i, T item) {
		item[convert(i)] = item;
	}
}

// In Java we cannot create an array of the generic type → cast
```

// **Minesweeper**

- Placing the bombs → place k bombs in the first k cells and shuffle the board
- Setting the numbered cells → go at each bomb location and increment each cell around it
- Expanding a blank region → use a queue

// **File System** → In-Memory file system

- contains Files and Directories
- recursive structure

// **Hash Table** → design a hash table that uses chaining to handle collisions