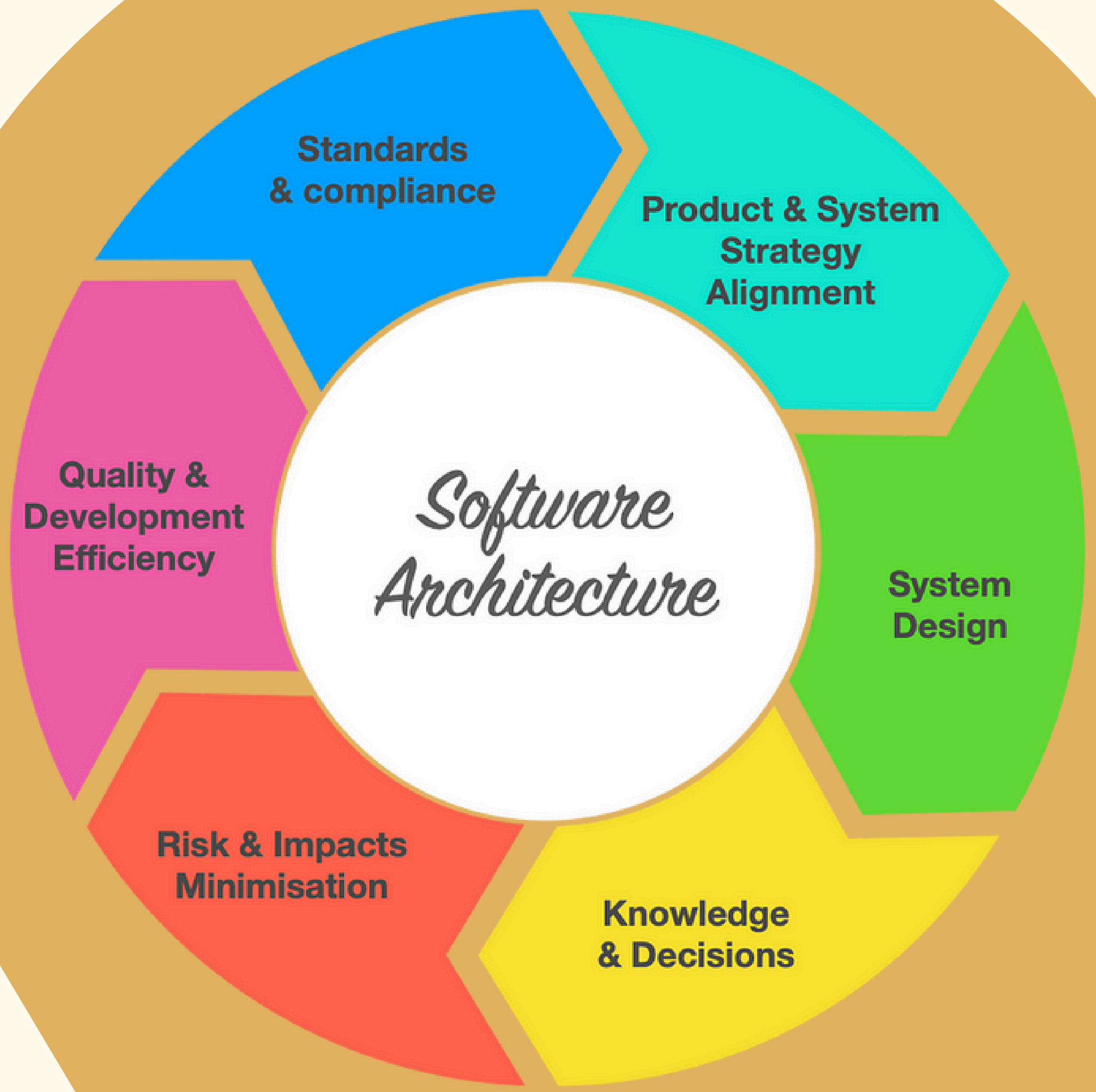
WEEK 8


Software Architecture
Standards & compliance
Product & System Strategy Alignment
System Design
Knowledge & Decisions
Risk & Impacts Minimisation
Quality & Development Efficiency

https://www.canva.com/design/DAGQDqrFOM0/tYcShPJMrfAxQmkaKsYQwg/edit

TASK 1

AGILITY AND MONOLITHIC ARCHITECTURES

MONOLITHIC ARCHITECTURES

- Simple
- inexpensive
- use when having tight budget and time constraints

SIMPLICITY

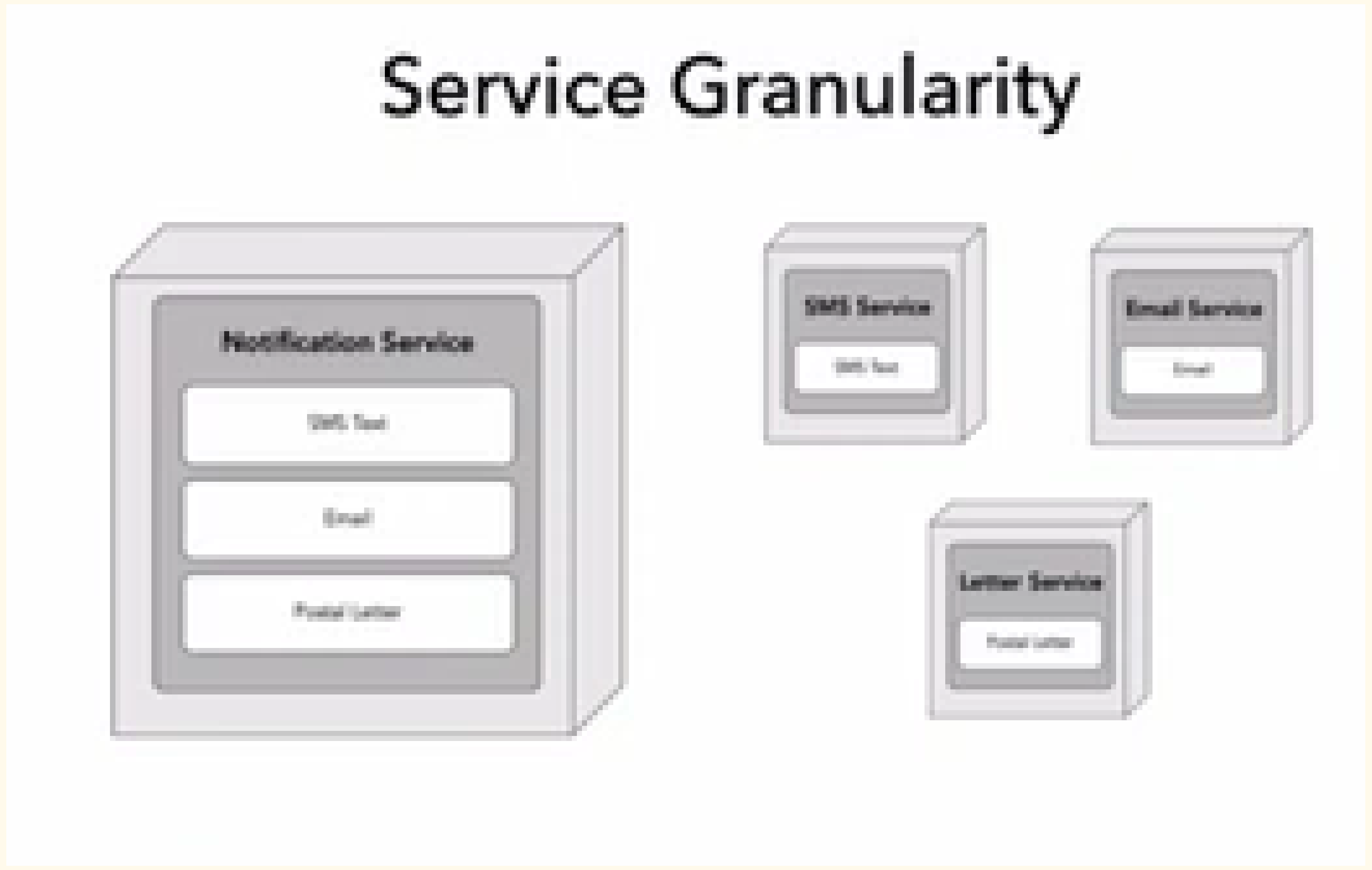
No need to worry about

- Service granularity
- Workflow coordination
- shared functionality
- communication protocols
- contract type
- Distributed data

(unlike distributed architecture)

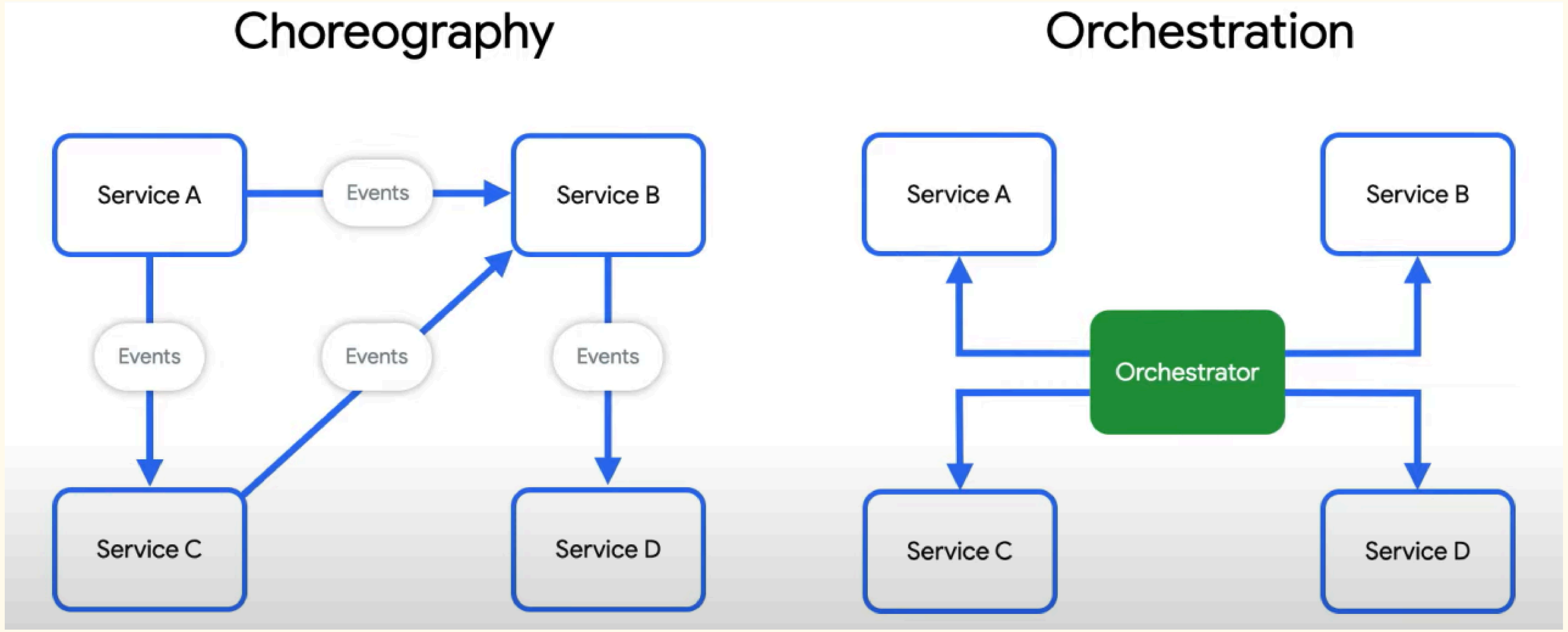
SERVICE GRANULARITY

- getting the size of the service correct
- how services are lumped together (coarse or fine graned



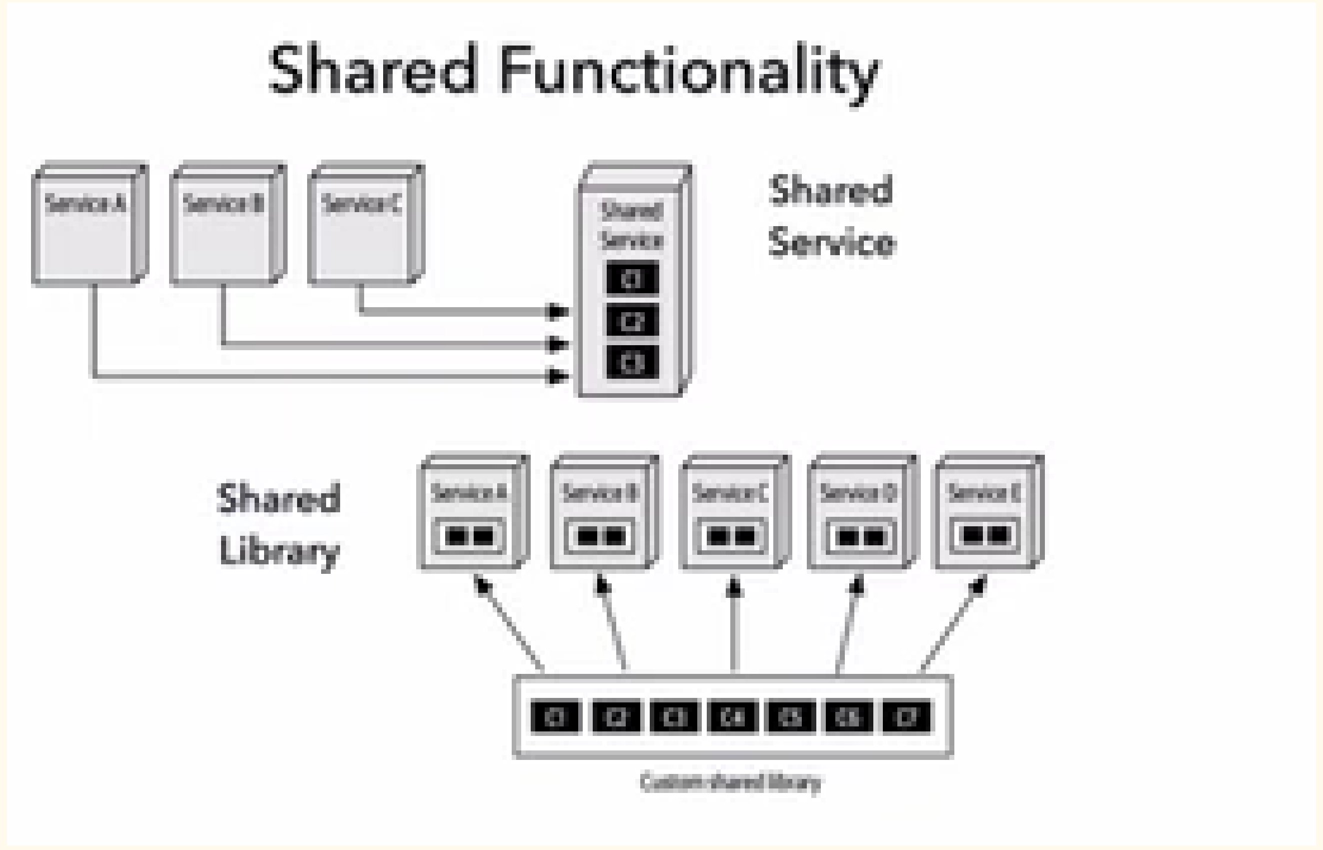
WORKFLOW COORDINATION

- whether to coordinate each services through orchestration or choreography



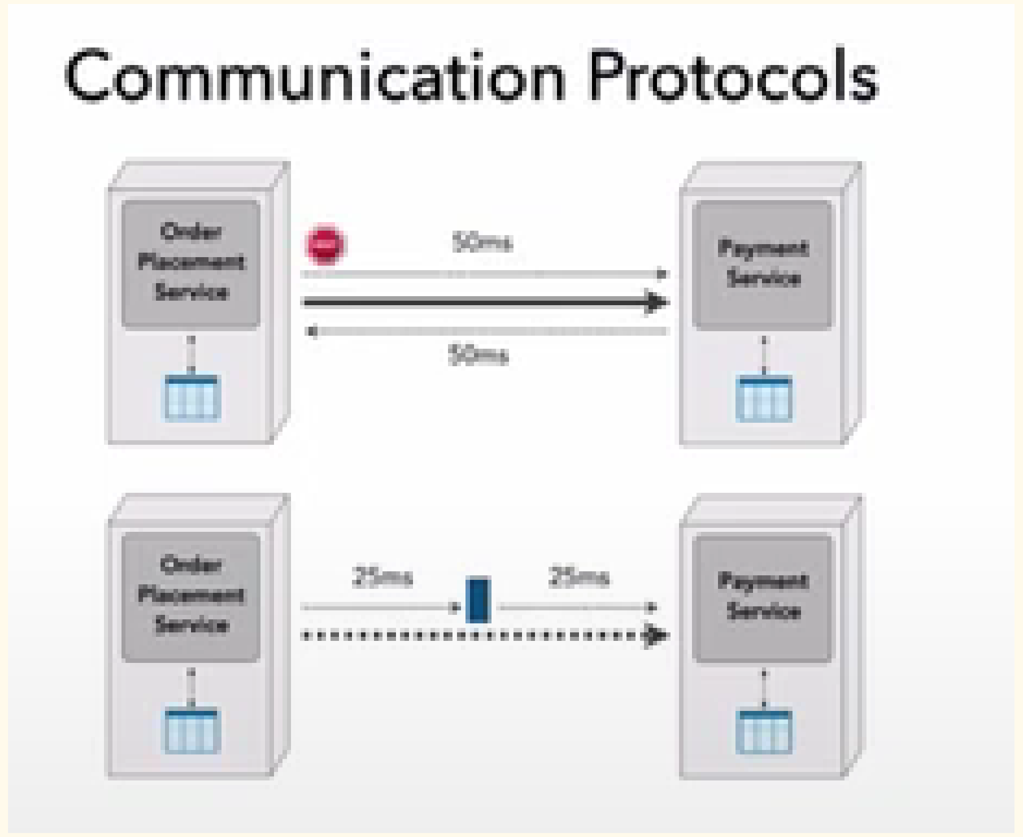
SHARED FUNCTIONALITY

- shared code or utility
- already packaged together in the monolithic architecture



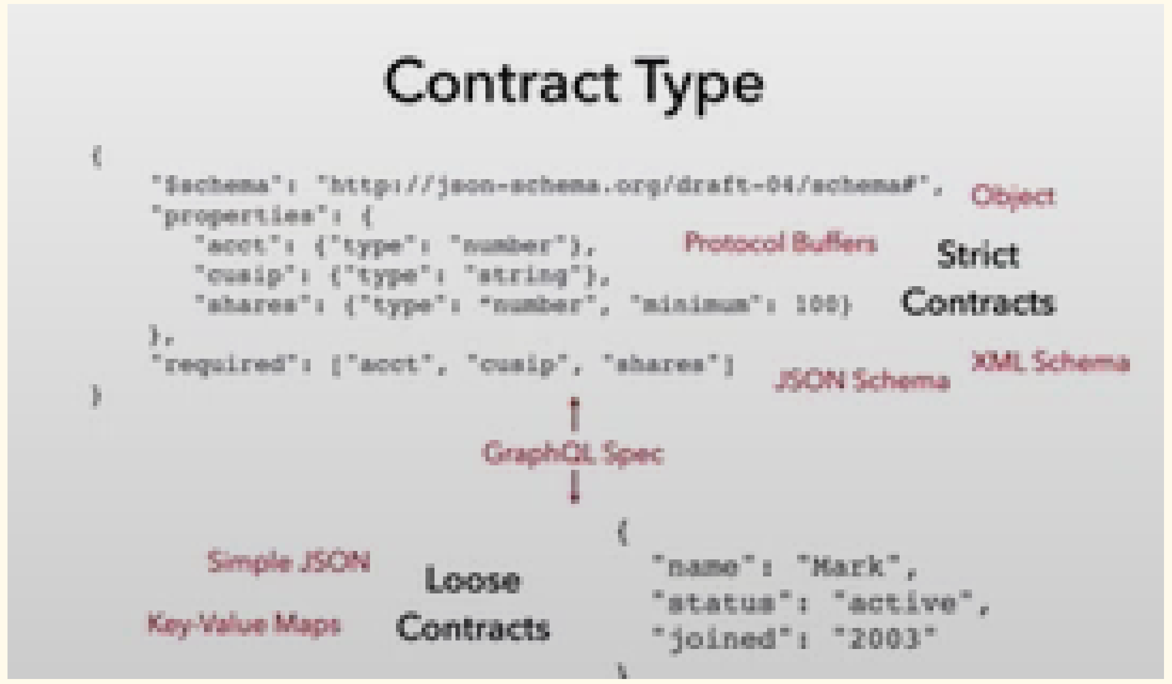
COMMUNICATION PROTOCOLS

- REST OR SOAP
- JSON OR XML
- HOW EACH SERVICE TALK TO EACH OTHER



CONTRACT TYPE

- will the rules be strict(xml schema) or loose (simple json) in the contract
- how different components or systems interact with each other

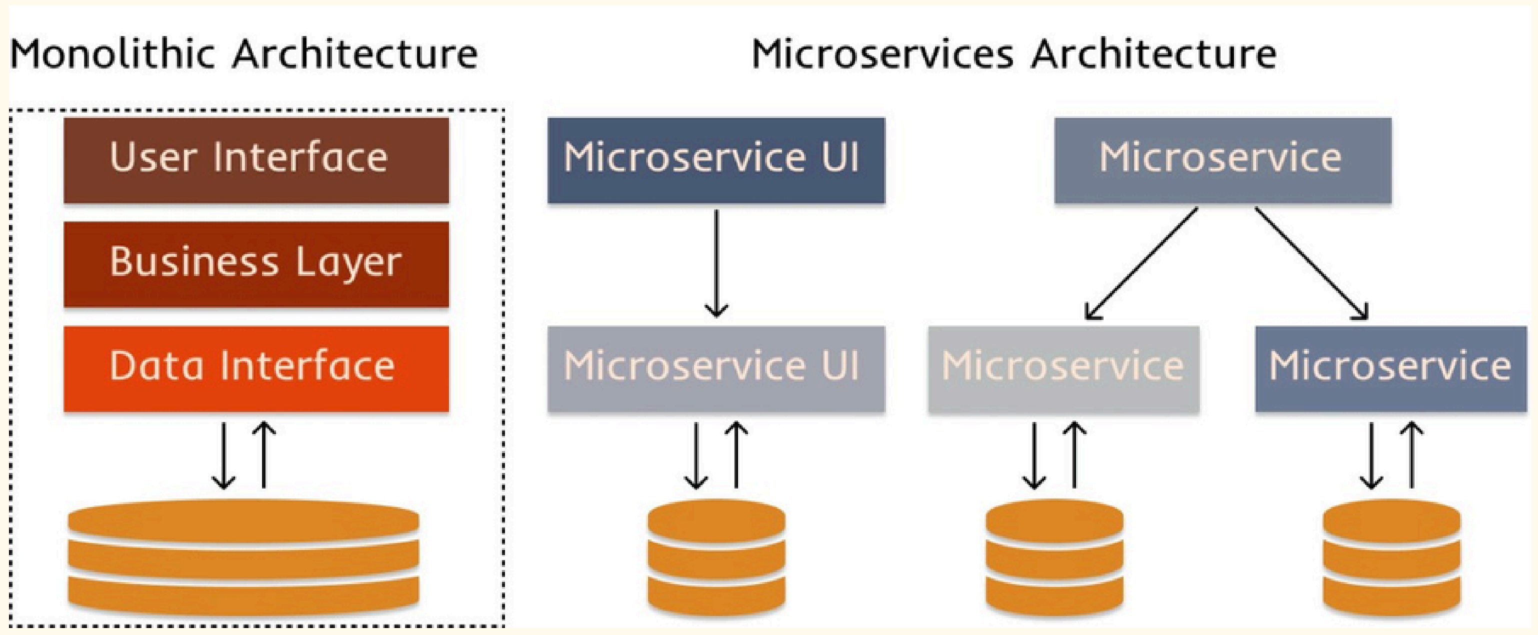


```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "properties": {
        "acct": {"type": "number"},
        "cusip": {"type": "string"},
        "shares": {"type": "number", "minimum": 100}
    },
    "required": ["acct", "cusip", "shares"]
}
```

```
{
  "name": "Mark",
  "status": "active",
  "joined": "2003"
```

DISTRIBUTED DATA

- how data are stored in the architecture
- distributed architecture has to think of having smaller storages for each service
- while monolithic simply has single storage / database shared across the system



AGILITY

“THE ABILITY TO RESPOND QUICKLY TO CHANGES”

MADE UP OF 3 THINGS

MAINTAINABILITY

ABILITY TO LOCATE WHERE YOU SHOULD

- APPLY CHANGES
- IDENTIFY AND FIX A BUG
- ADDING FEATURES

TESTABILITY

- EASE OF TESTING
- COMPLETENESS OF TESTING

DEPLOYABILITY

- FREQUENCY OF DEPLOYMENT
- OVERALL RISKS

WHEN NOT TO USE MONOLITHIC ARCHITECTURE

EVERYBODY IS MAKING THE CHANGE TO THE SAME CODE BASE

WHEN NOT TO USE MONOLITHIC ARCHITECTURE

MAKE THE MAINTAINABILITY DIFFICULT

- CHANGING ONE PLACE MIGHT AFFECT OTHER COMPONENTS

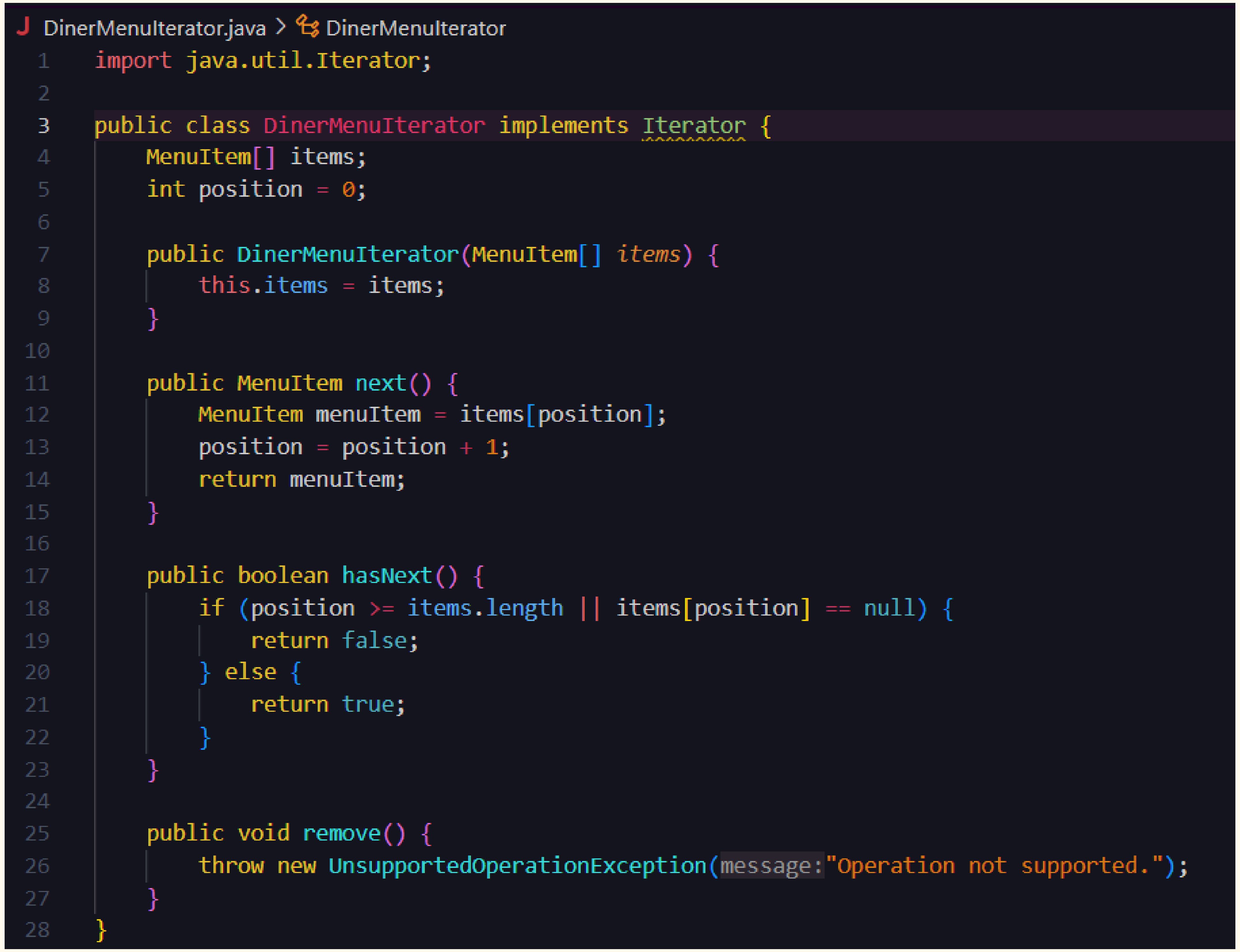
WHEN NOT TO USE MONOLITHIC ARCHITECTURE

SCOPE OF THE SYSTEM IS TOO LARGE

- DON’T KNOW WHERE THE ERROR STARTS
- DIFFICULTY IN COMPLETENESS OF TESTING
    - CHANGES AREN’T COORDINATED
- CODE FREEZES IMPACT DEPLOYABILITY

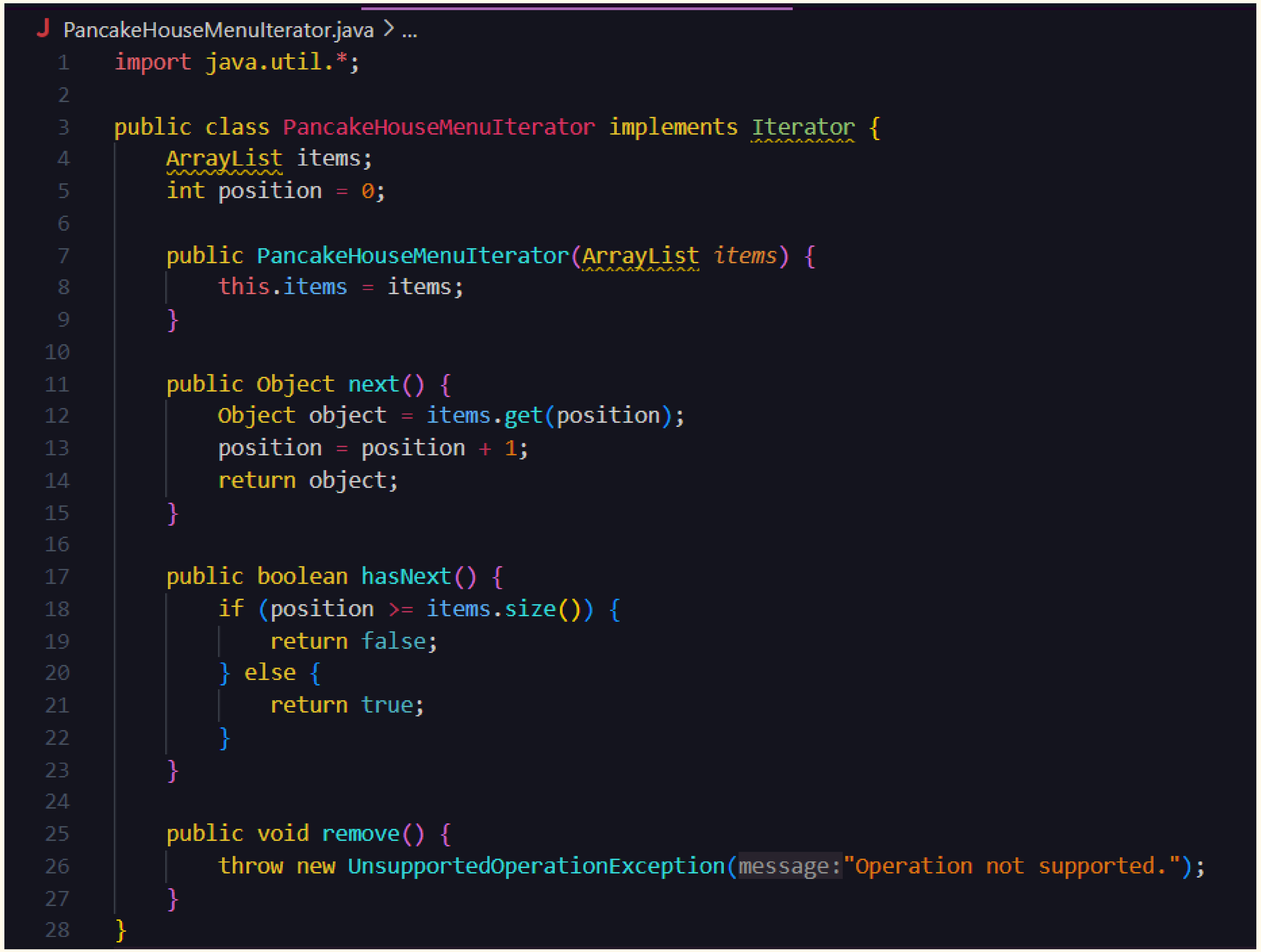
TASK 2

DinerMenuIterator.java


DinerMenuIterator.java > DinerMenuIterator

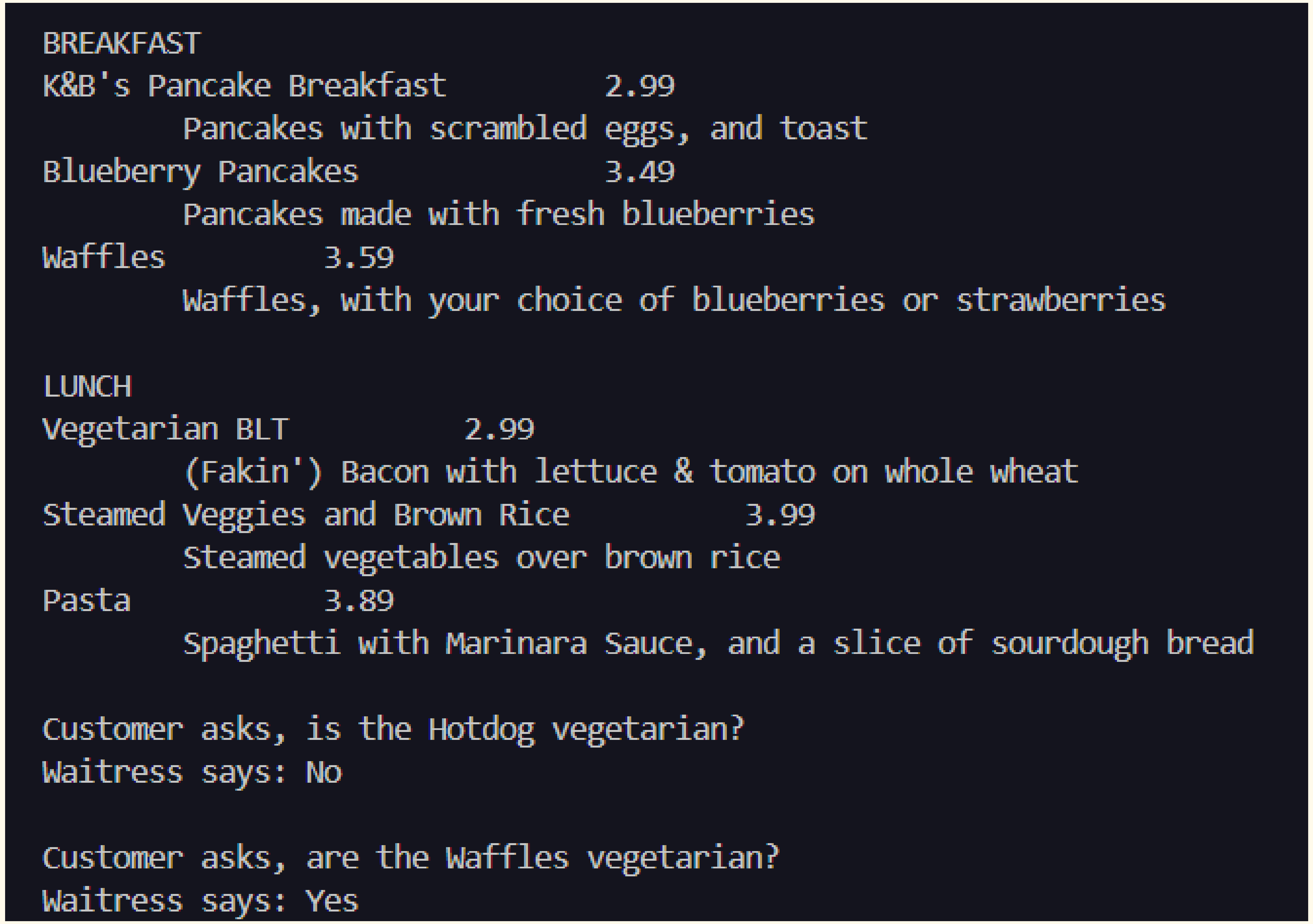
```java
import java.util.Iterator;

public class DinerMenuIterator implements Iterator {
    MenuItem[] items;
    int position = 0;

    public DinerMenuIterator(MenuItem[] items) {
        this.items = items;
    }

    public MenuItem next() {
        MenuItem menuItem = items[position];
        position = position + 1;
        return menuItem;
    }

    public boolean hasNext() {
        if (position >= items.length || items[position] == null) {
            return false;
        } else {
            return true;
        }
    }

    public void remove() {
        throw new UnsupportedOperationException(message:"Operation not supported.");
    }
}
```

PancakeHouseMenuIterator.java



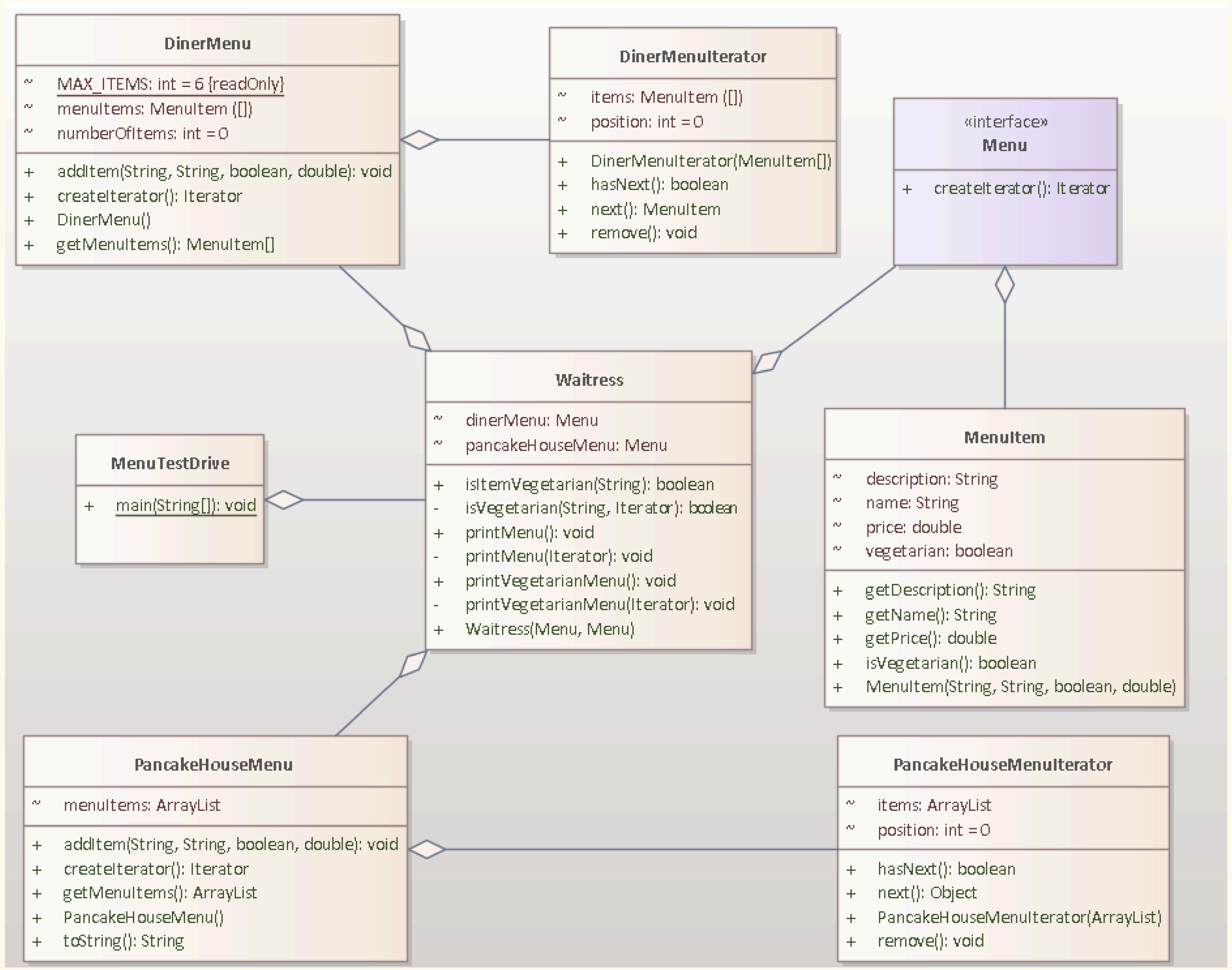
PancakeHouseMenuIterator.java > ...

```java
import java.util.*;

public class PancakeHouseMenuIterator implements Iterator {
    ArrayList items;
    int position = 0;

    public PancakeHouseMenuIterator(ArrayList items) {
        this.items = items;
    }

    public Object next() {
        Object object = items.get(position);
        position = position + 1;
        return object;
    }

    public boolean hasNext() {
        if (position >= items.size()) {
            return false;
        } else {
            return true;
        }
    }

    public void remove() {
        throw new UnsupportedOperationException(message:"Operation not supported.");
    }
}
```



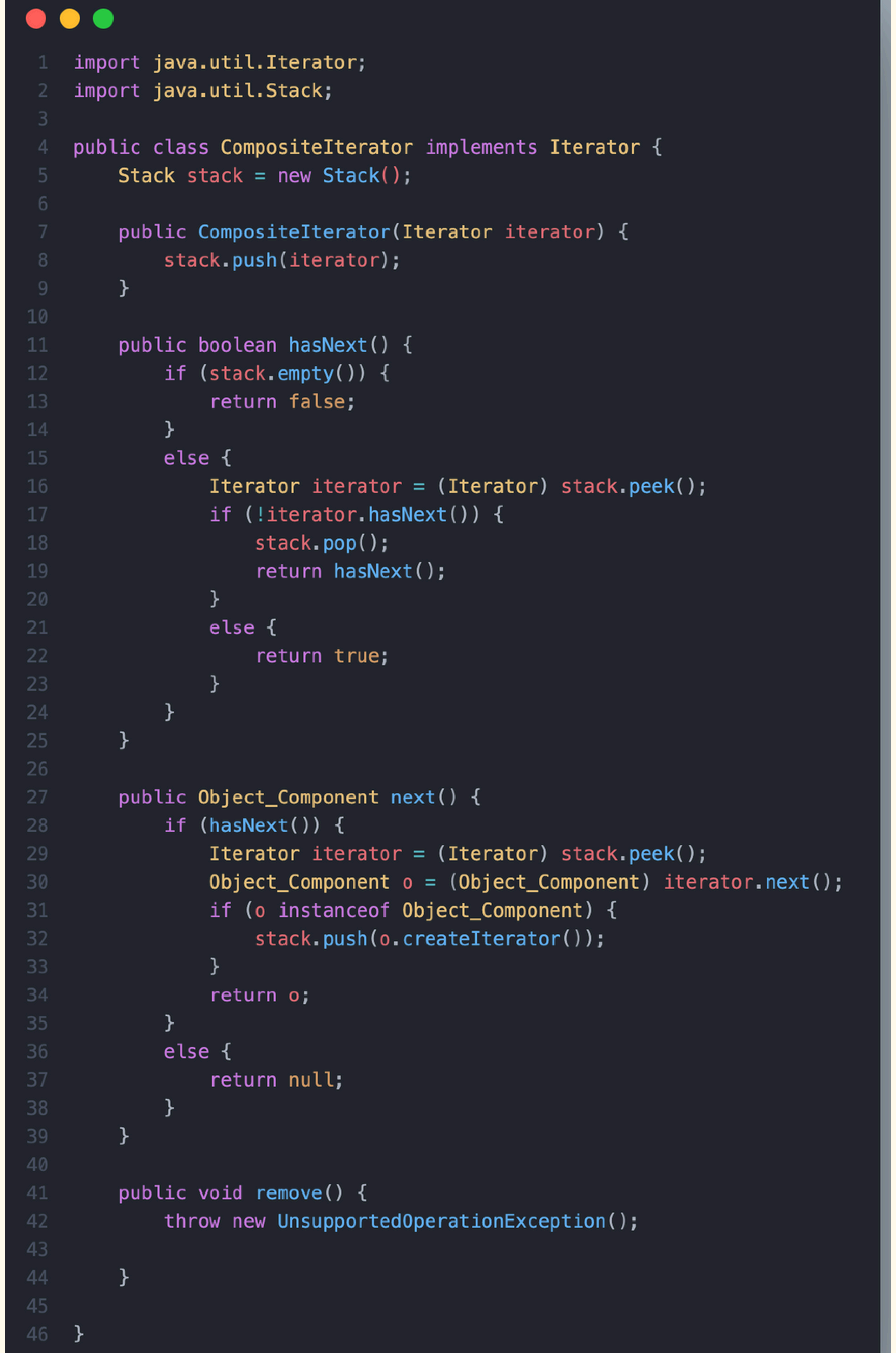
```
BREAKFAST
K&B's Pancake Breakfast         2.99
        Pancakes with scrambled eggs, and toast
Blueberry Pancakes              3.49
        Pancakes made with fresh blueberries
Waffles         3.59
        Waffles, with your choice of blueberries or strawberries

LUNCH
Vegetarian BLT          2.99
        (Fakin') Bacon with lettuce & tomato on whole wheat
Steamed Veggies and Brown Rice          3.99
        Steamed vegetables over brown rice
Pasta           3.89
        Spaghetti with Marinara Sauce, and a slice of sourdough bread

Customer asks, is the Hotdog vegetarian?
Waitress says: No

Customer asks, are the Waffles vegetarian?
Waitress says: Yes
```

CLASS DIAGRAM

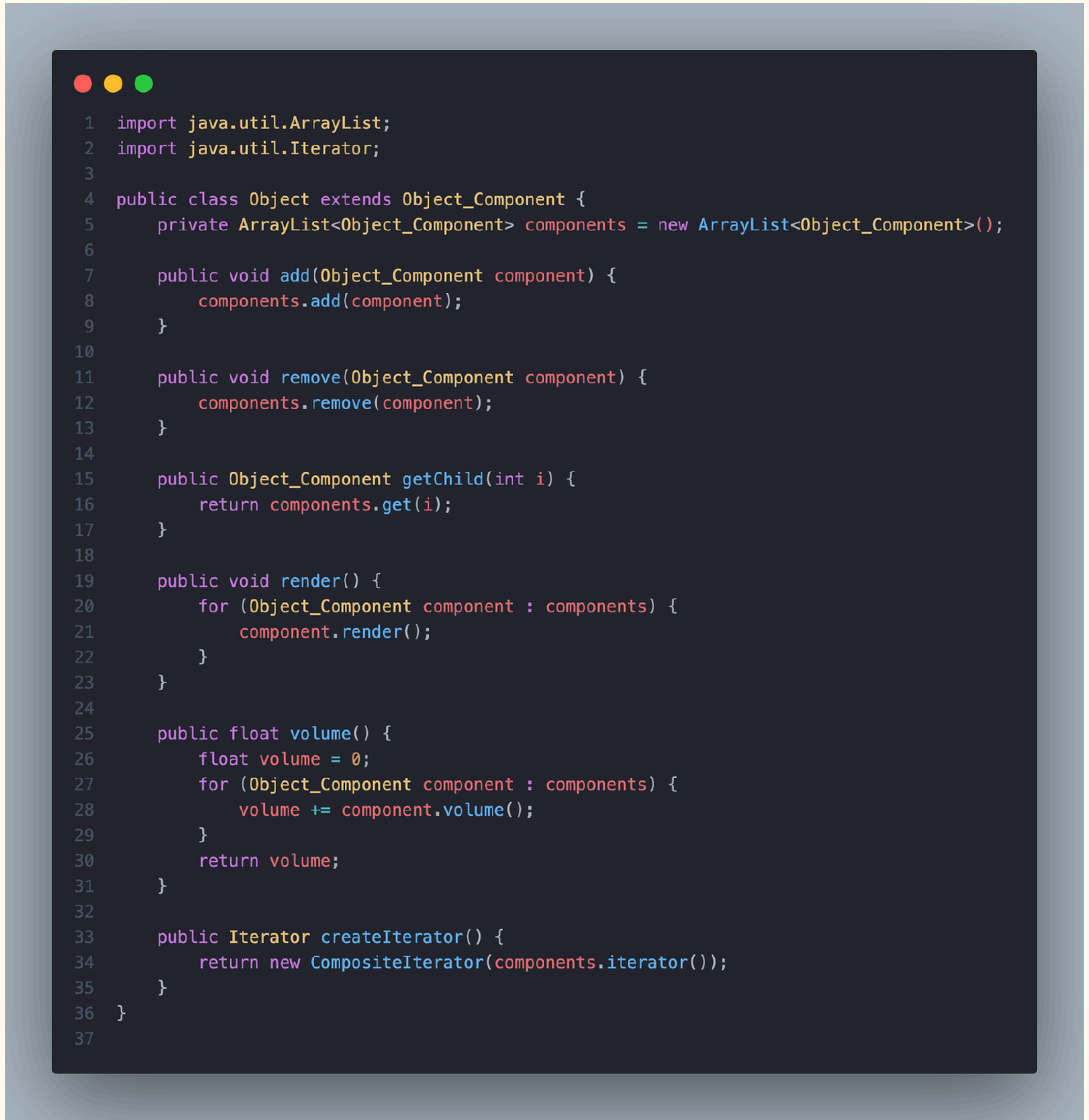

DinerMenu
~ MAX_ITEMS: int = 6 {readOnly}
~ menuItems: MenuItem ([])
~ numberOfItems: int = 0
+ addItem(String, String, boolean, double): void
+ createIterator(): Iterator
+ DinerMenu()
+ getMenuItems(): MenuItem[]
DinerMenuIterator
~ items: MenuItem ([])
~ position: int = 0
+ DinerMenuIterator(MenuItem[])
+ hasNext(): boolean
+ next(): MenuItem
+ remove(): void
«interface»
Menu
+ createIterator(): Iterator
MenuTestDrive
+ main(String[]): void
Waitress
~ dinerMenu: Menu
~ pancakeHouseMenu: Menu
+ isItemVegetarian(String): boolean
- isVegetarian(String, Iterator): boolean
+ printMenu(): void
- printMenu(Iterator): void
+ printVegetarianMenu(): void
- printVegetarianMenu(Iterator): void
+ Waitress(Menu, Menu)
MenuItem
~ description: String
~ name: String
~ price: double
~ vegetarian: boolean
+ getDescription(): String
+ getName(): String
+ getPrice(): double
+ isVegetarian(): boolean
+ MenuItem(String, String, boolean, double)
PancakeHouseMenu
~ menuItems: ArrayList
+ addItem(String, String, boolean, double): void
+ createIterator(): Iterator
+ getMenuItems(): ArrayList
+ PancakeHouseMenu()
+ toString(): String
PancakeHouseMenuIterator
~ items: ArrayList
~ position: int = 0
+ hasNext(): boolean
+ next(): Object
+ PancakeHouseMenuIterator(ArrayList)
+ remove(): void

TASK 3

compositeIterator.java

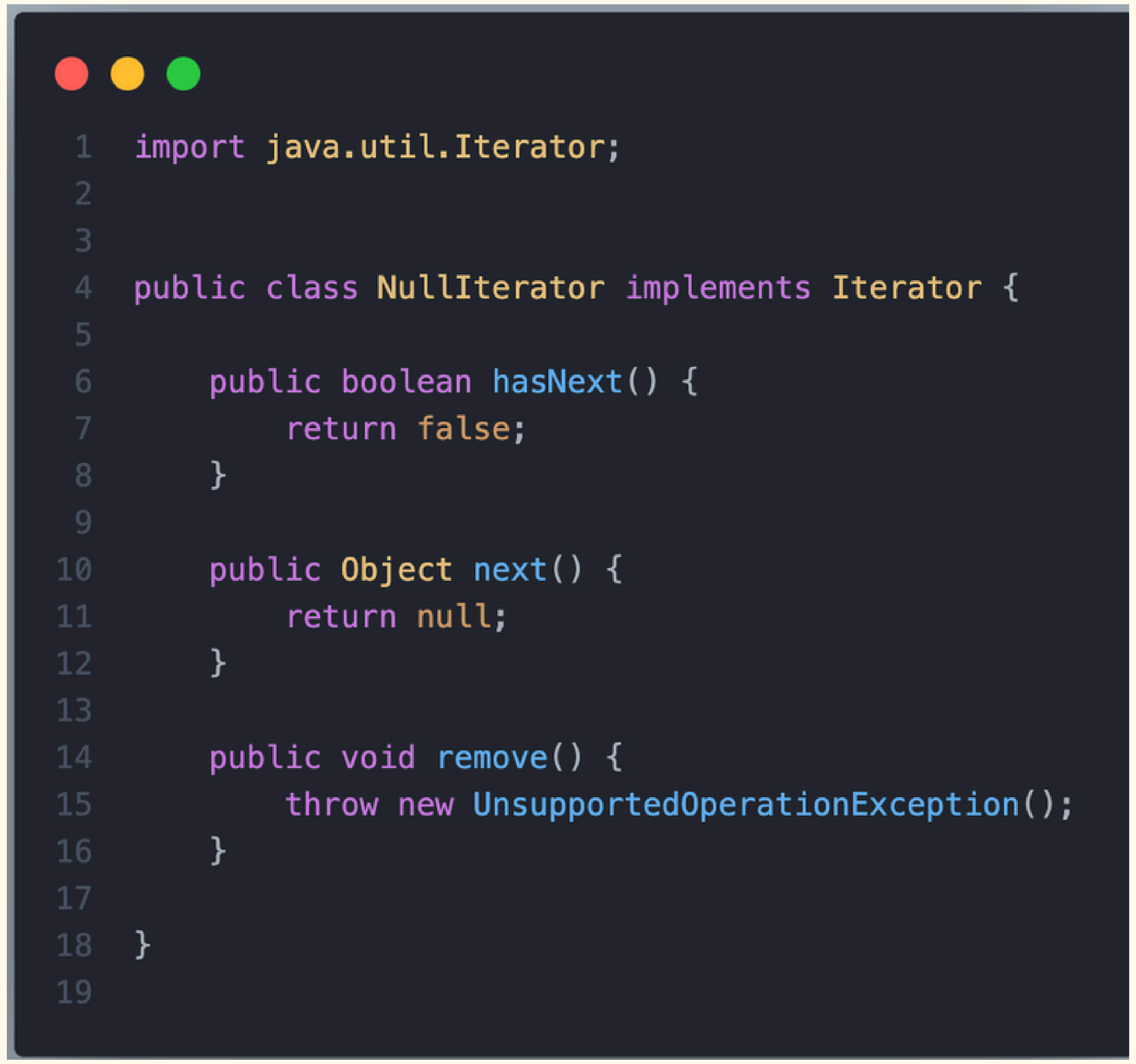

```java
import java.util.Iterator;
import java.util.Stack;

public class CompositeIterator implements Iterator {
    Stack stack = new Stack();

    public CompositeIterator(Iterator iterator) {
        stack.push(iterator);
    }

    public boolean hasNext() {
        if (stack.empty()) {
            return false;
        }
        else {
            Iterator iterator = (Iterator) stack.peek();
            if (!iterator.hasNext()) {
                stack.pop();
                return hasNext();
            }
            else {
                return true;
            }
        }
    }

    public Object_Component next() {
        if (hasNext()) {
            Iterator iterator = (Iterator) stack.peek();
            Object_Component o = (Object_Component) iterator.next();
            if (o instanceof Object_Component) {
                stack.push(o.createIterator());
            }
            return o;
        }
        else {
            return null;
        }
    }

    public void remove() {
        throw new UnsupportedOperationException();

    }

}
```

Object.java

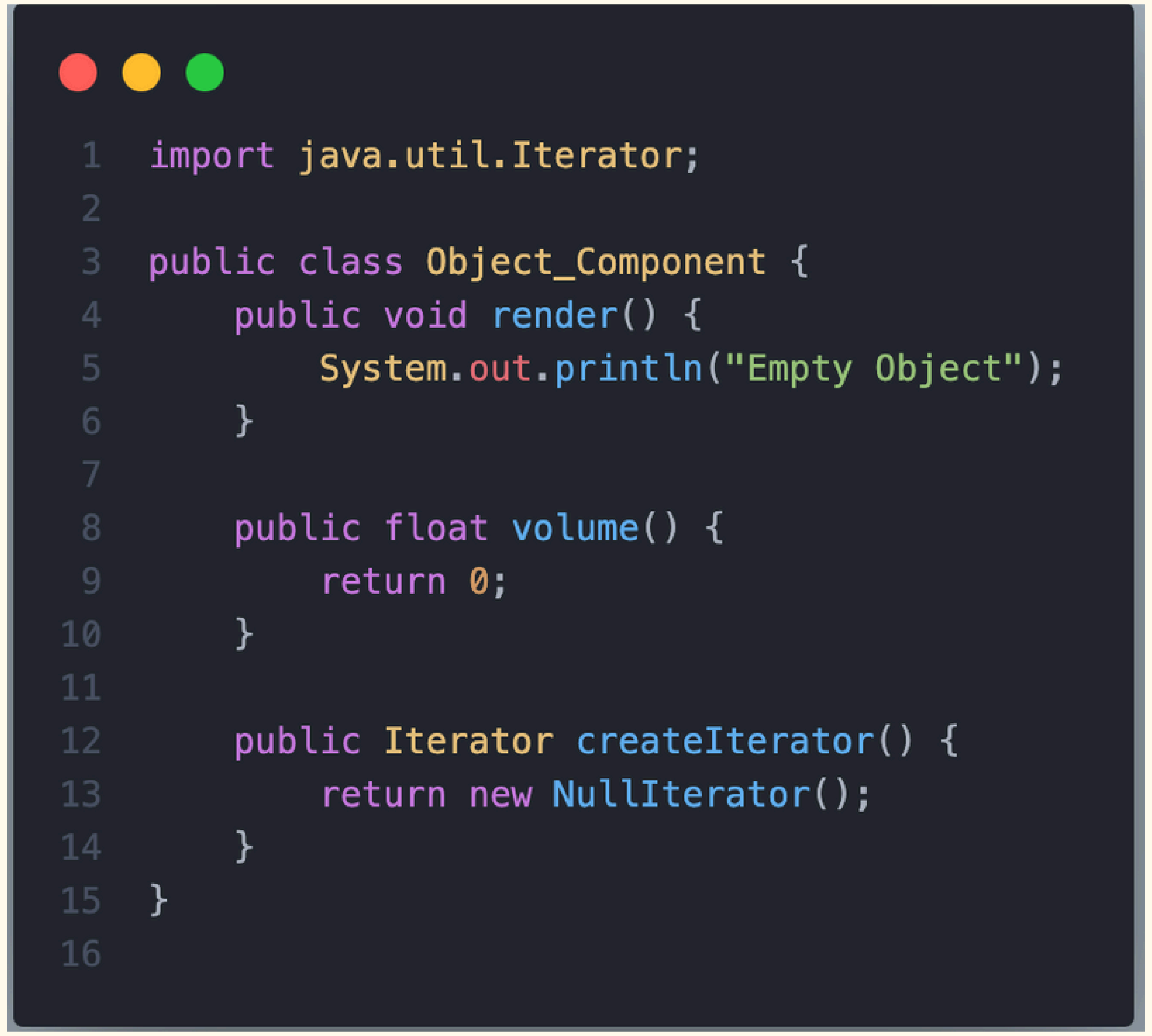


```java
import java.util.ArrayList;
import java.util.Iterator;

public class Object extends Object_Component {
    private ArrayList<Object_Component> components = new ArrayList<Object_Component>();

    public void add(Object_Component component) {
        components.add(component);
    }

    public void remove(Object_Component component) {
        components.remove(component);
    }

    public Object_Component getChild(int i) {
        return components.get(i);
    }

    public void render() {
        for (Object_Component component : components) {
            component.render();
        }
    }

    public float volume() {
        float volume = 0;
        for (Object_Component component : components) {
            volume += component.volume();
        }
        return volume;
    }

    public Iterator createIterator() {
        return new CompositeIterator(components.iterator());
    }
}

```

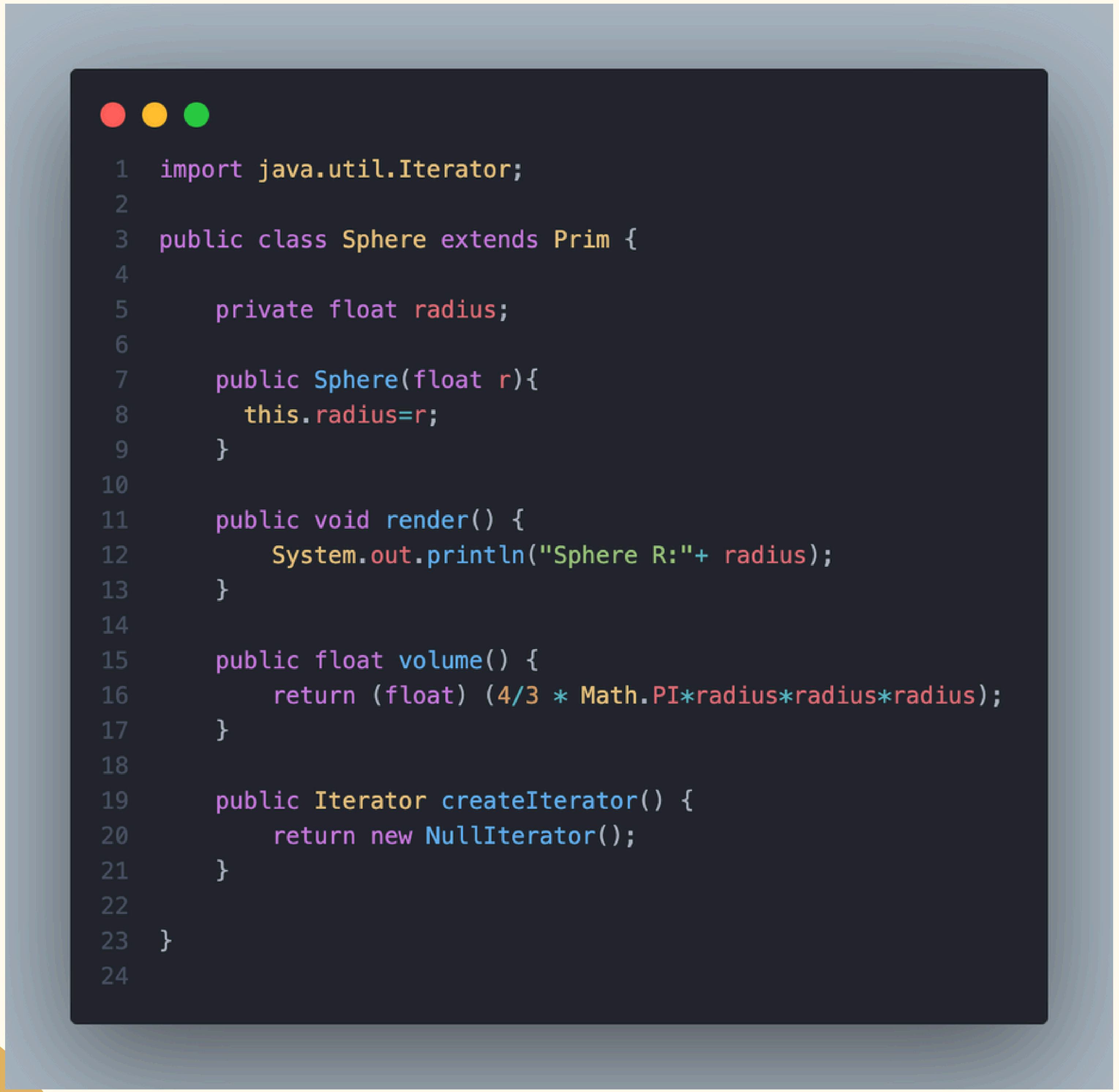
NullIterator.java


```java
import java.util.Iterator;


public class NullIterator implements Iterator {

    public boolean hasNext() {
        return false;
    }

    public Object next() {
        return null;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

}

```

Object_Component.java

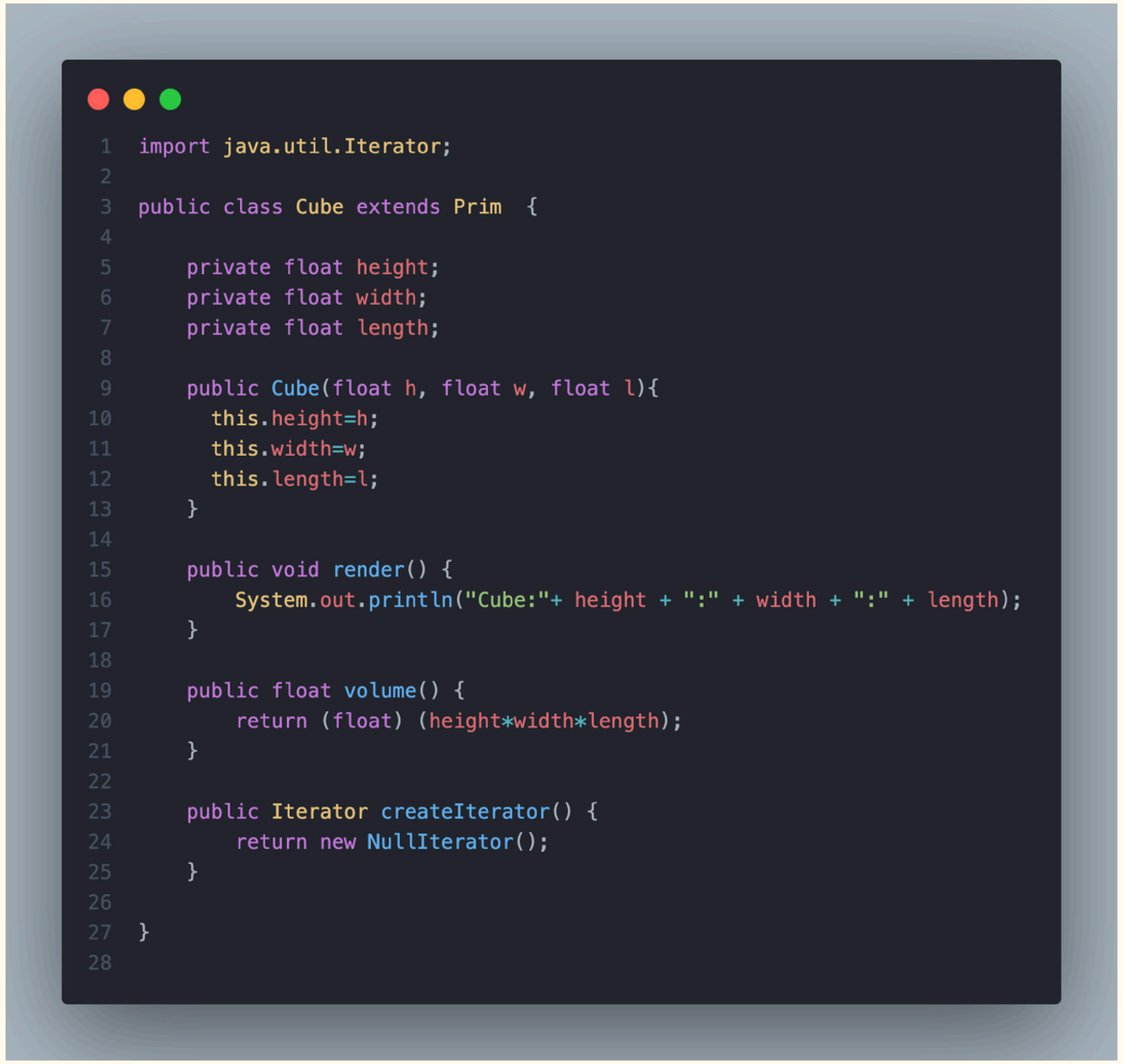

```java
import java.util.Iterator;

public class Object_Component {
    public void render() {
        System.out.println("Empty Object");
    }

    public float volume() {
        return 0;
    }

    public Iterator createIterator() {
        return new NullIterator();
    }
}

```

Sphere.java



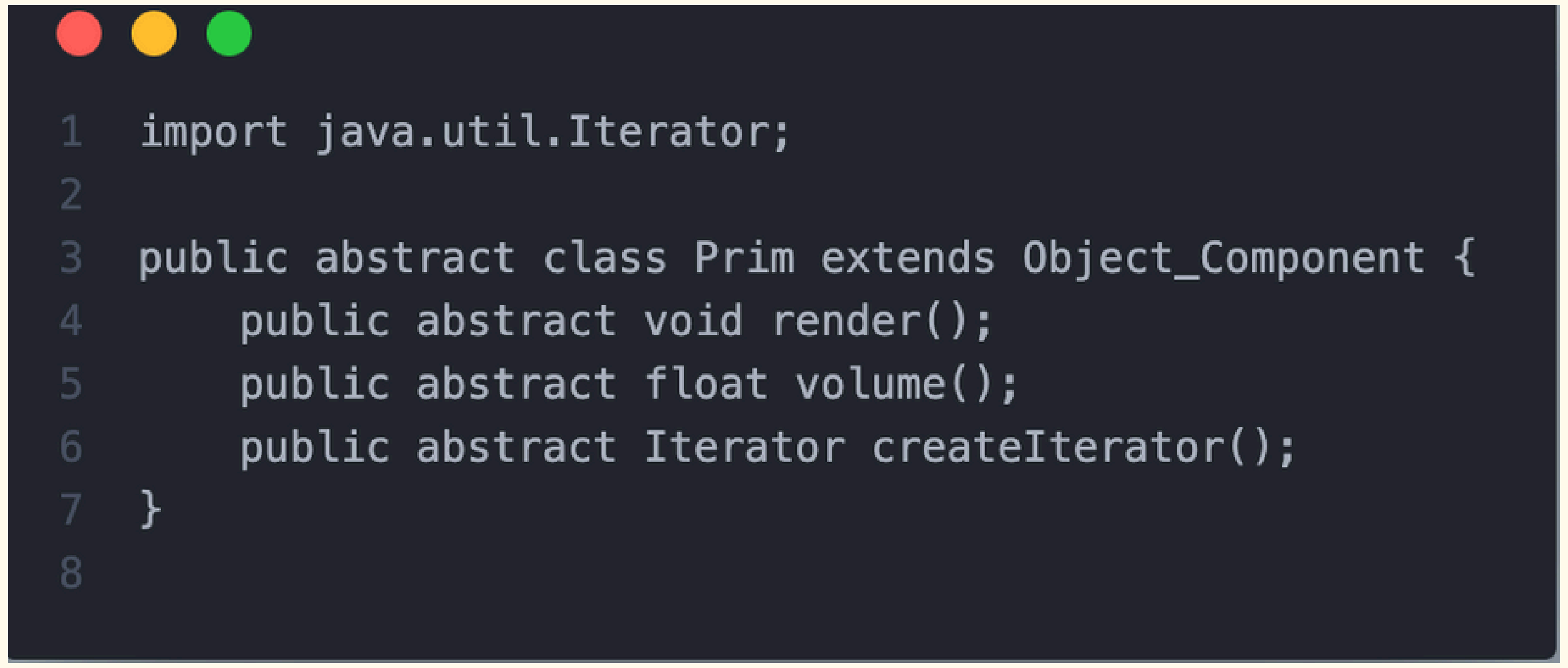
```java
import java.util.Iterator;

public class Sphere extends Prim {

    private float radius;

    public Sphere(float r){
      this.radius=r;
    }

    public void render() {
        System.out.println("Sphere R:"+ radius);
    }

    public float volume() {
        return (float) (4/3 * Math.PI*radius*radius*radius);
    }

    public Iterator createIterator() {
        return new NullIterator();
    }

}
```

Cube.java



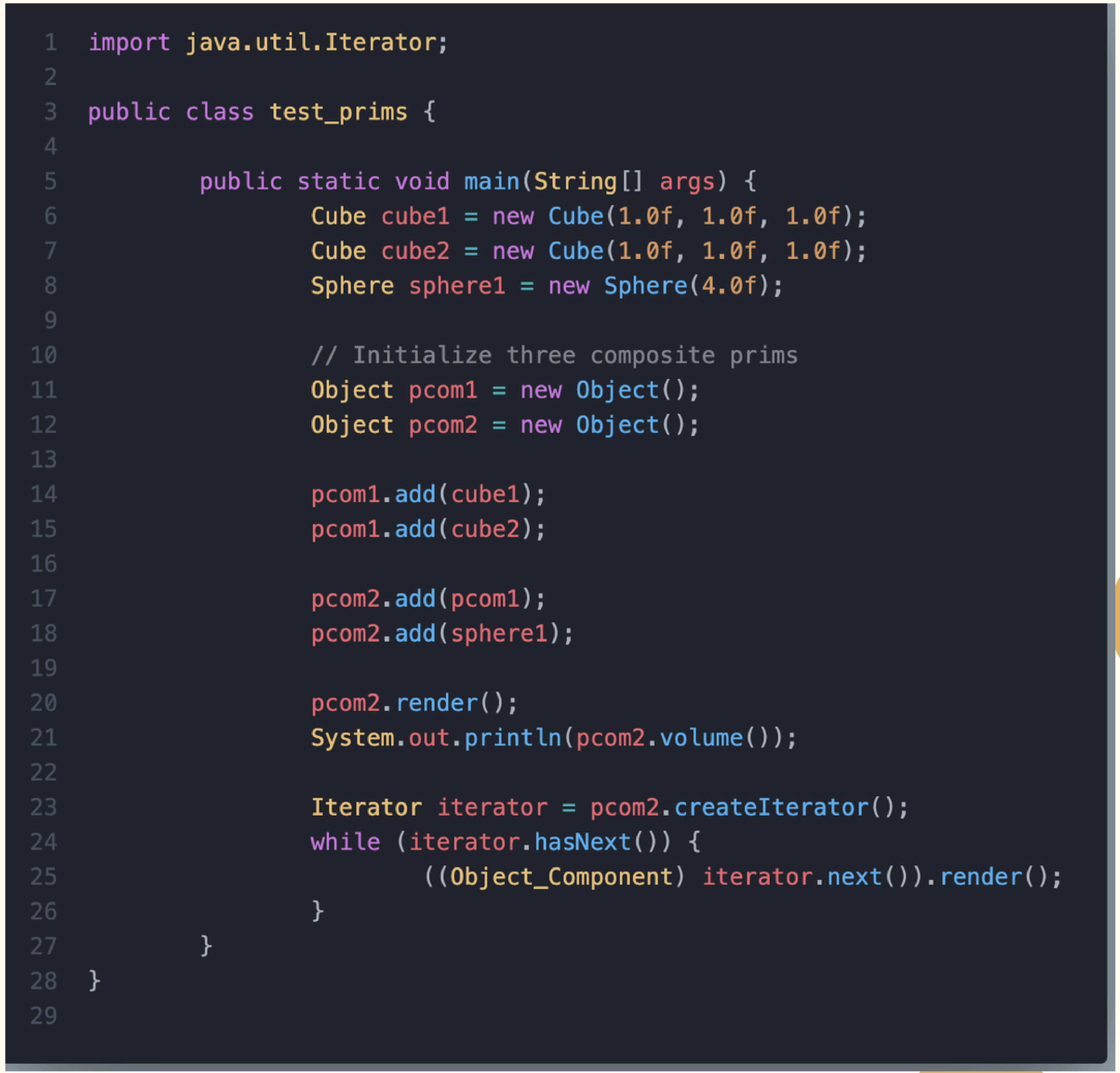
```java
import java.util.Iterator;

public class Cube extends Prim  {

    private float height;
    private float width;
    private float length;

    public Cube(float h, float w, float l){
      this.height=h;
      this.width=w;
      this.length=l;
    }

    public void render() {
        System.out.println("Cube:"+ height + ":" + width + ":" + length);
    }

    public float volume() {
        return (float) (height*width*length);
    }

    public Iterator createIterator() {
        return new NullIterator();
    }

}
```

prim.java



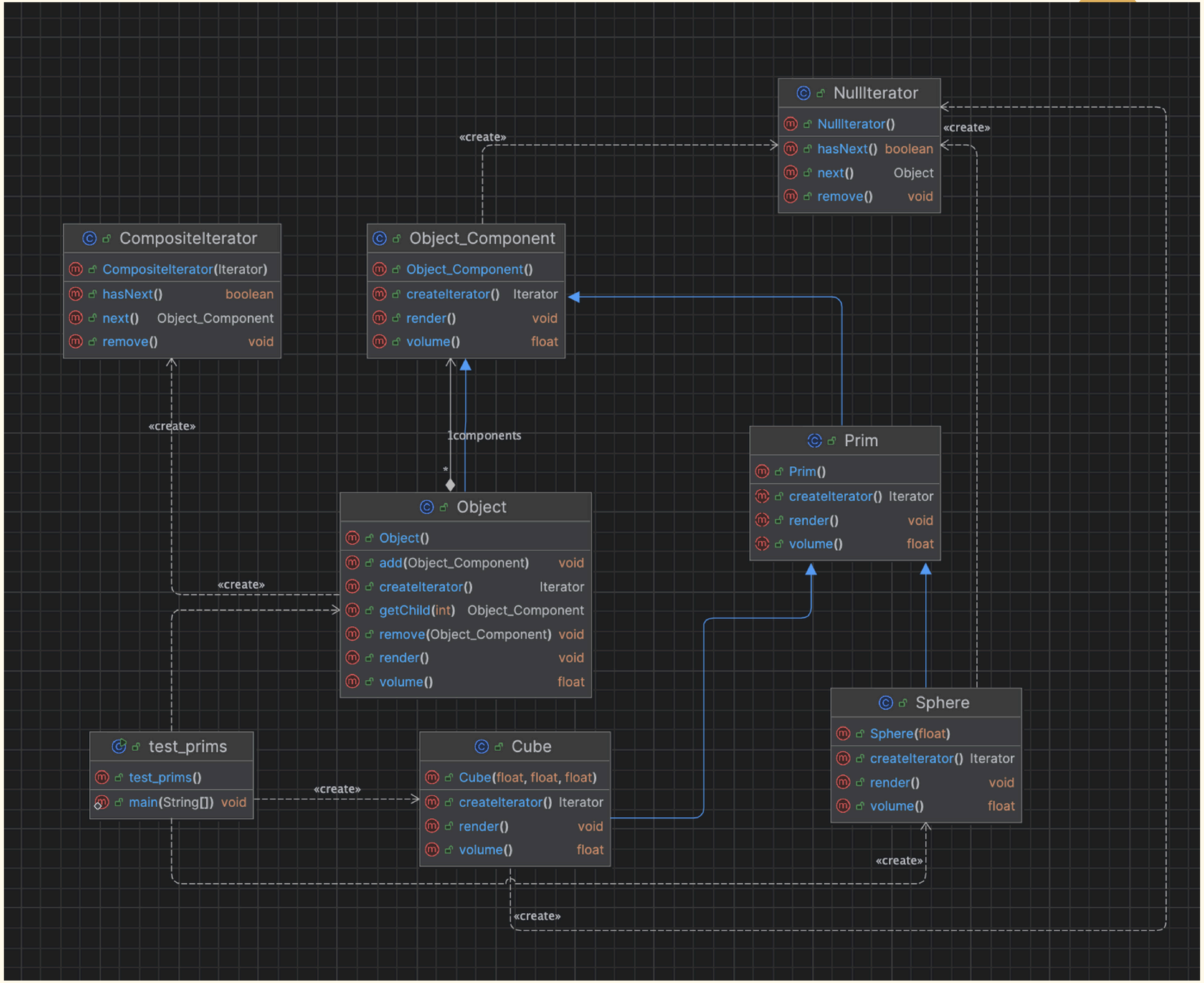
```java
import java.util.Iterator;

public abstract class Prim extends Object_Component {
    public abstract void render();
    public abstract float volume();
    public abstract Iterator createIterator();
}

```

test_prims.java

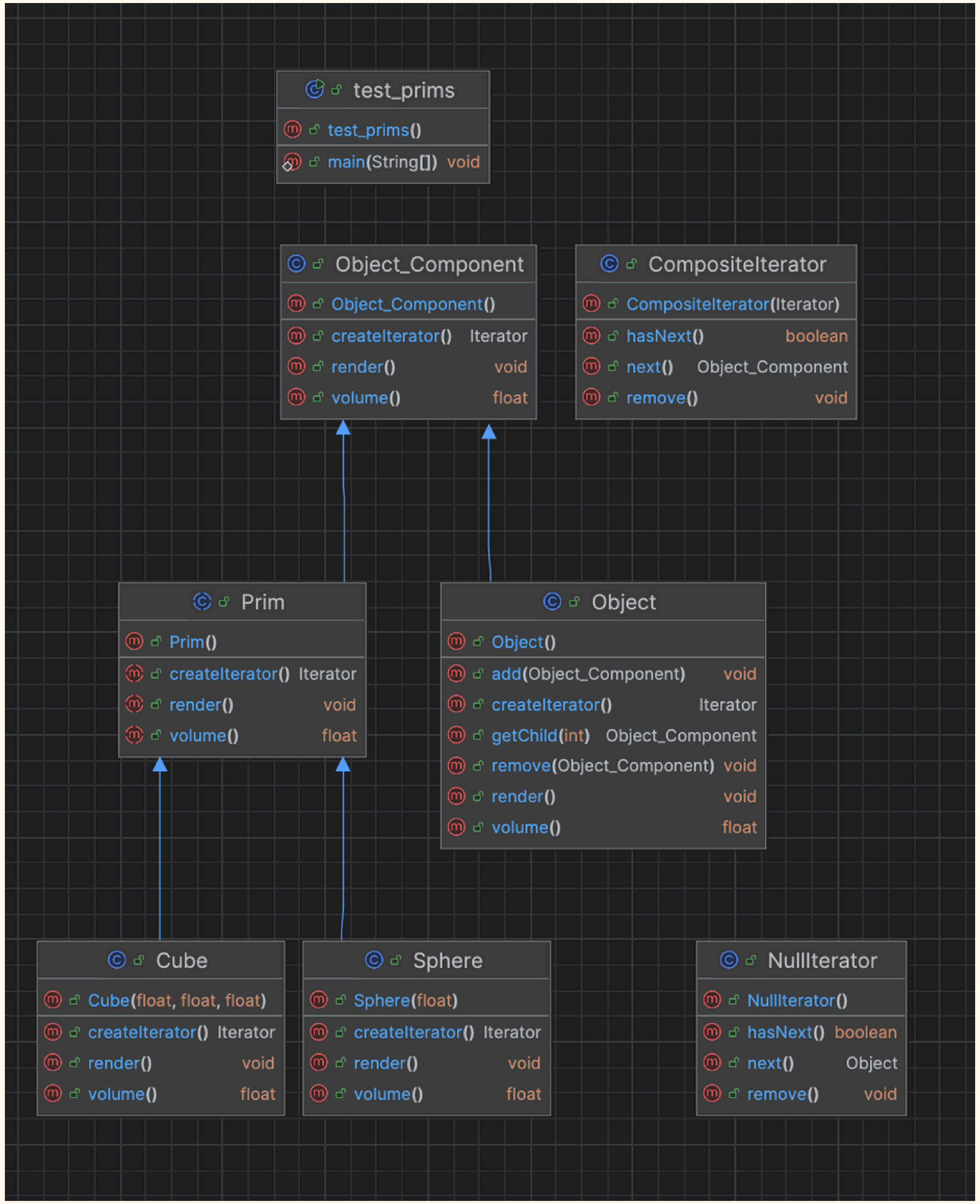


```java
import java.util.Iterator;

public class test_prims {

        public static void main(String[] args) {
                Cube cube1 = new Cube(1.0f, 1.0f, 1.0f);
                Cube cube2 = new Cube(1.0f, 1.0f, 1.0f);
                Sphere sphere1 = new Sphere(4.0f);

                // Initialize three composite prims
                Object pcom1 = new Object();
                Object pcom2 = new Object();

                pcom1.add(cube1);
                pcom1.add(cube2);

                pcom2.add(pcom1);
                pcom2.add(sphere1);

                pcom2.render();
                System.out.println(pcom2.volume());

                Iterator iterator = pcom2.createIterator();
                while (iterator.hasNext()) {
                        ((Object_Component) iterator.next()).render();
                }
        }
}

```

CLASS DIAGRAM



CLASS DIAGRAM



PRESENTED BY

65011277 Chanasorn Howattanakulphong

65011320 Kanokjan Singhsuwan

65011381 Napatr Sapprasert

65011400 Natthawut Lin

65011462 Phupa Denphatcharangkul

65011558 Suvijuk Samitimata

65011572 Teerapat Senanuch

THANK YOU