# Advanced Software Engineering Part 05 - Coupling and Cohesion

Dr. Amjad AbuHassan

# Modules Coupling

# Introduction to Coupling

- Modules coupling refers to the degree to which one module depends on another module.
- In software engineering, coupling is the measure of how closely connected two modules are.
- High coupling between modules can make the system more complex and difficult to maintain.

# Types of Coupling

There are different types of coupling, each representing a different level of dependency between modules:

- Content coupling
- Common coupling
- Control coupling
- Stamp coupling
- Data coupling

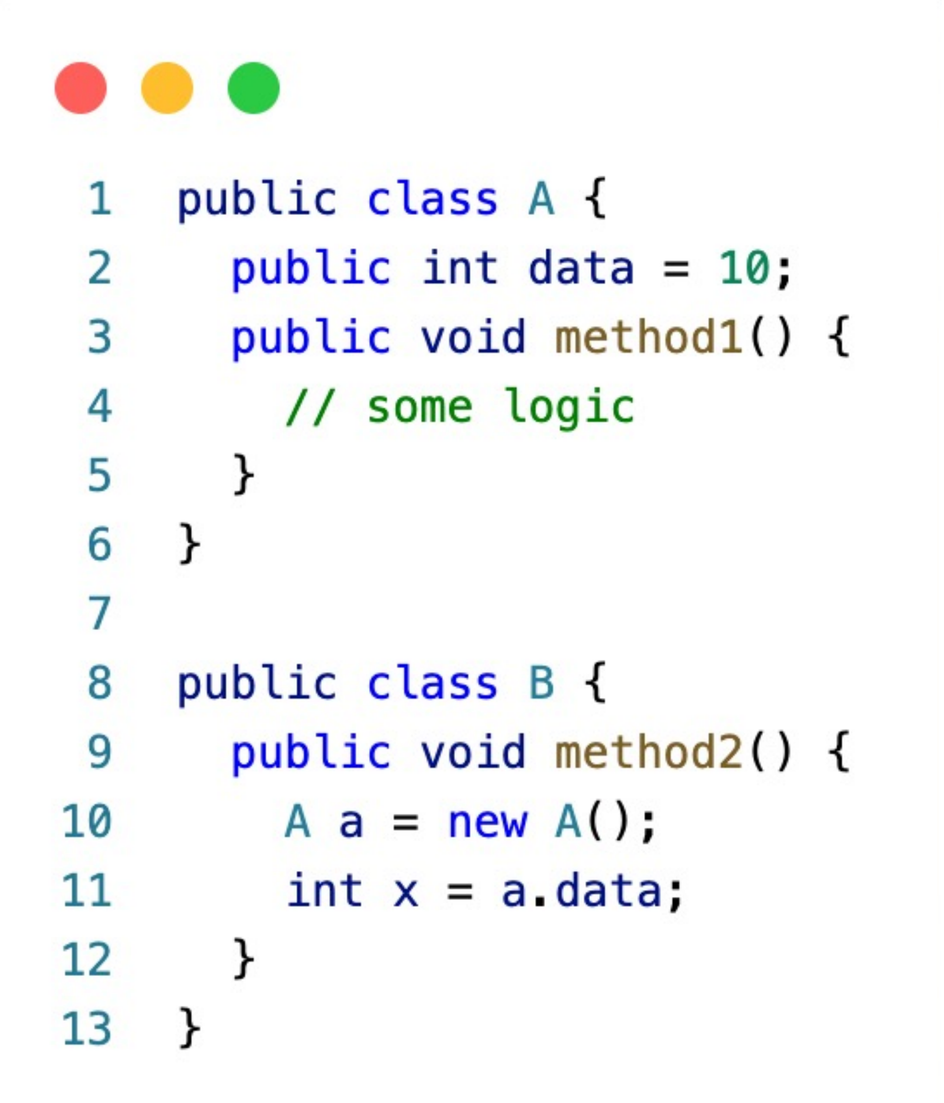
# Content Coupling

- One module directly accesses or modifies the content of another module.
- Occurs when one module depends on the internal workings of another module, such as by accessing its data structures or calling its private methods.

# Content Coupling cont.

- This is the tightest form of coupling and should be avoided whenever possible because it can make the code difficult to maintain and change.
- If one module's implementation details change, it can have a ripple effect throughout the codebase.

# Example



```java
public class A {
  public int data = 10;
  public void method1() {
    // some logic
  }
}

public class B {
  public void method2() {
    A a = new A();
    int x = a.data;
  }
}
```

# Common Coupling

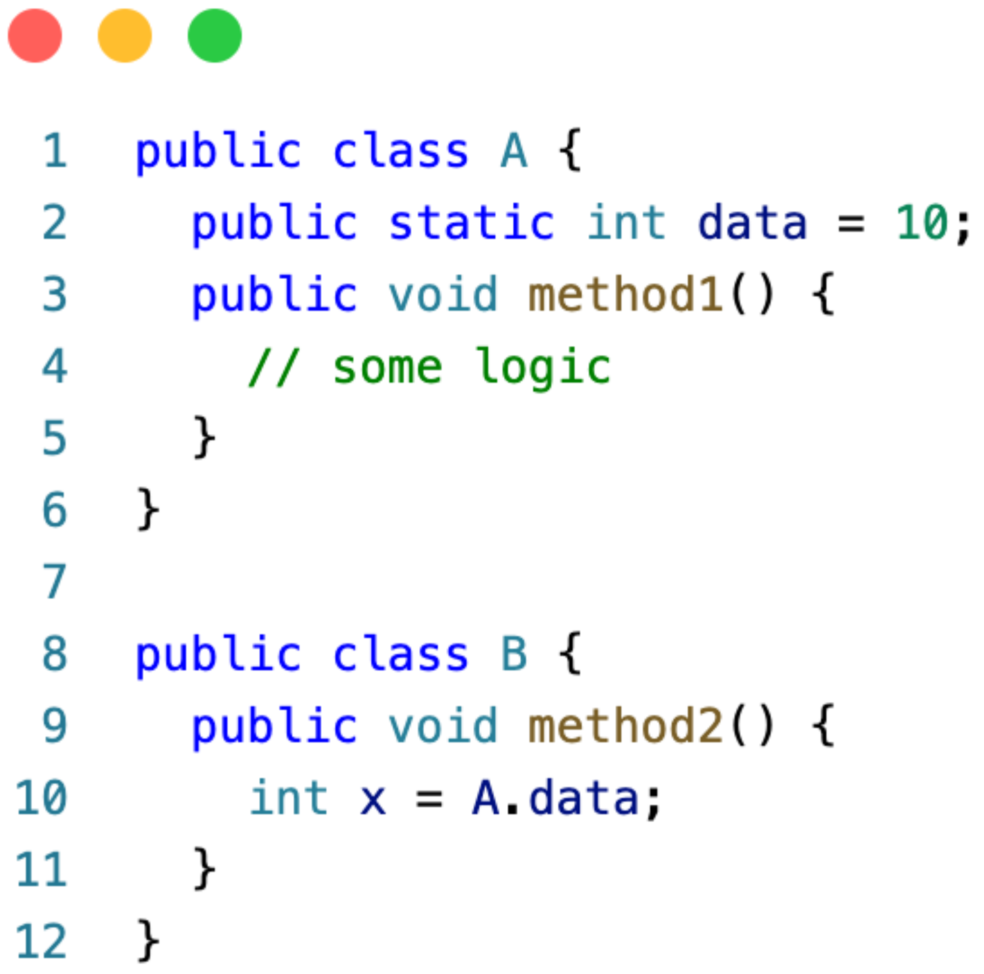
- Common coupling occurs when two or more modules share a common data source.

- Modules communicate by accessing a global data area that is accessible to all modules.

# Common Coupling cont.

- This type of coupling can make it difficult to maintain and test the system because changes in one module can affect other modules.
- Common coupling can be useful when multiple modules need to access the same data source or resource.
- However, it can also lead to problems if the shared resource is changed or becomes unavailable.

# Example



```
public class A {
  public static int data = 10;
  public void method1() {
    // some logic
  }
}

public class B {
  public void method2() {
    int x = A.data;
  }
}
```

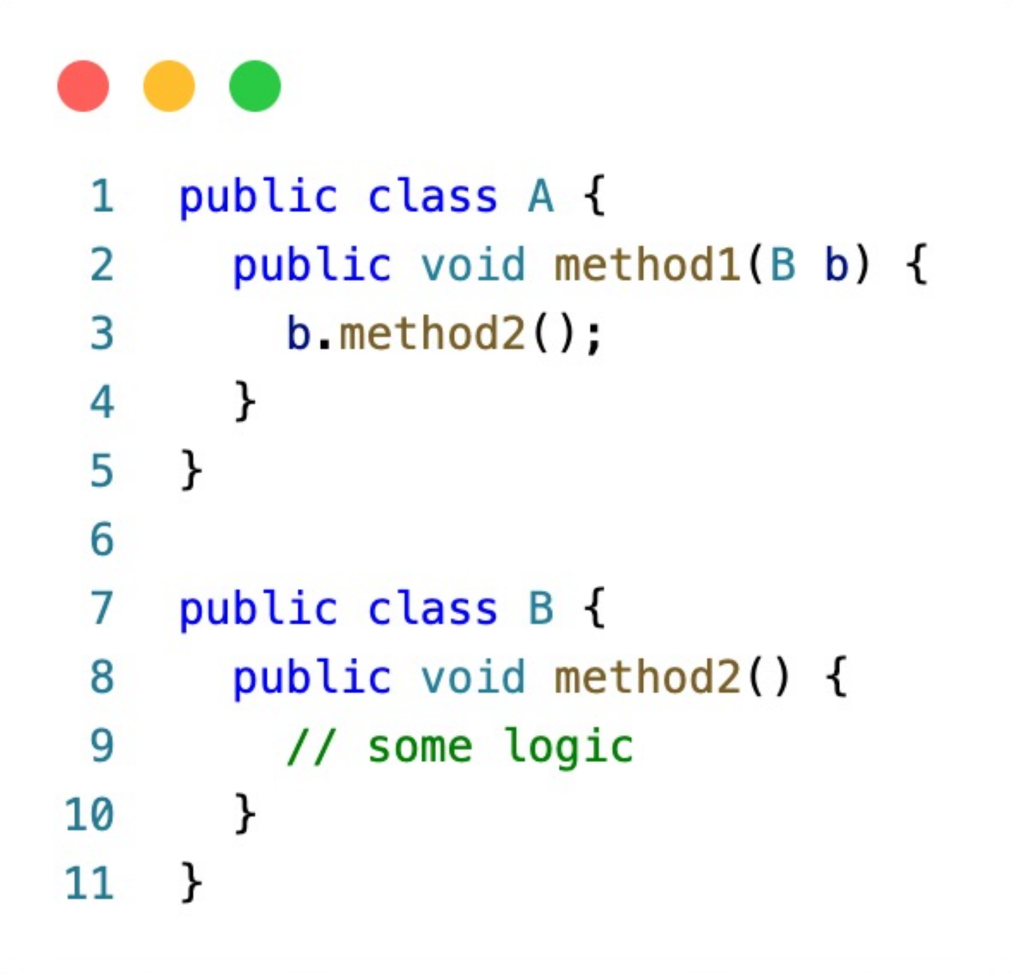
# Control Coupling

- Control coupling occurs when one module controls the behavior of another module.
- This happens by passing control information to another module such as flags, enums, switches, parameters, or other control information to another module
  - A module controls the flow of another module.

# Control Coupling cont.

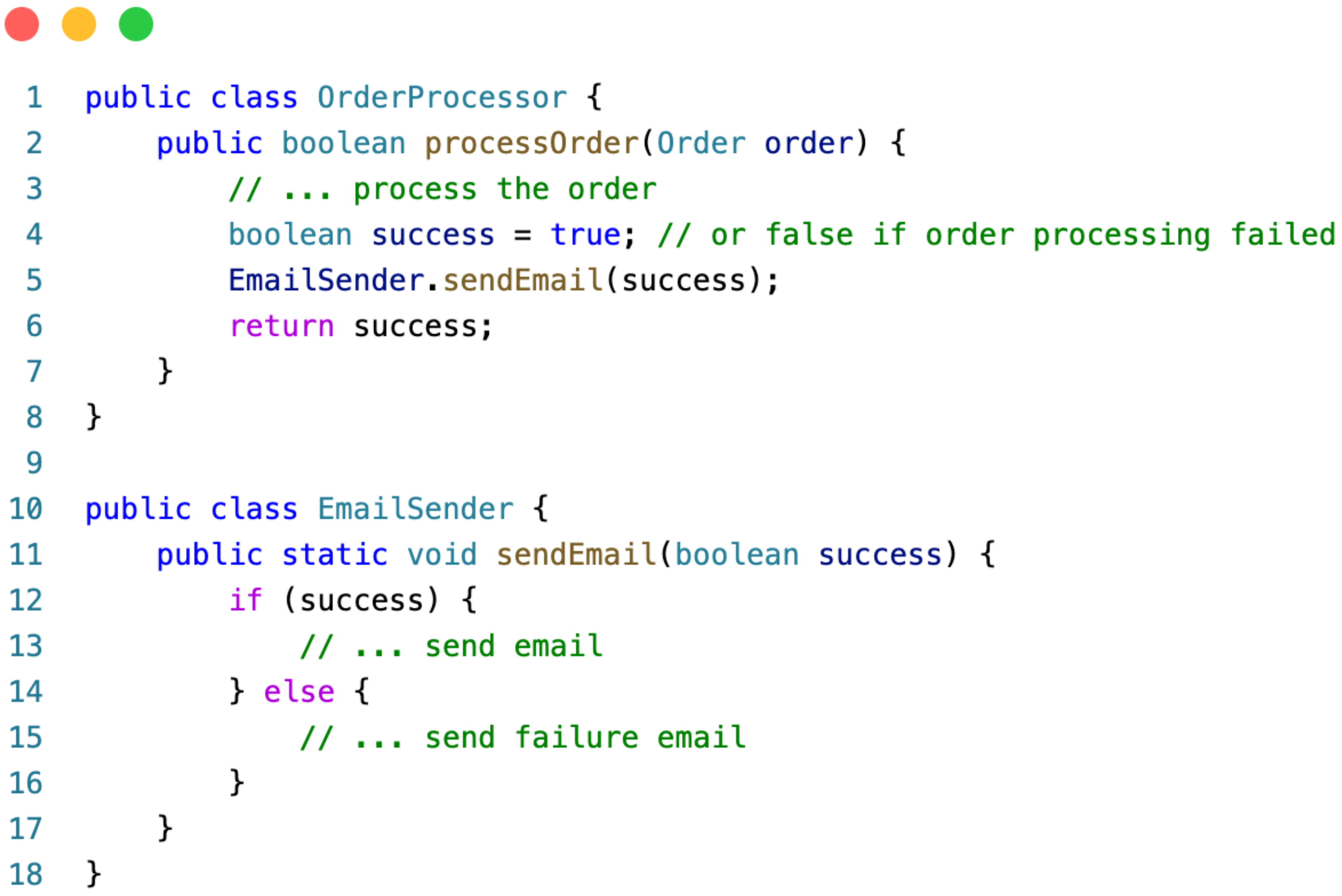
- The receiving module then takes different actions based on the control information.
- This type of coupling can make the system more difficult to understand and test.

# Example 1



```
public class A {
  public void method1(B b) {
    b.method2();
  }
}

public class B {
  public void method2() {
    // some logic
  }
}
```

# Example 2

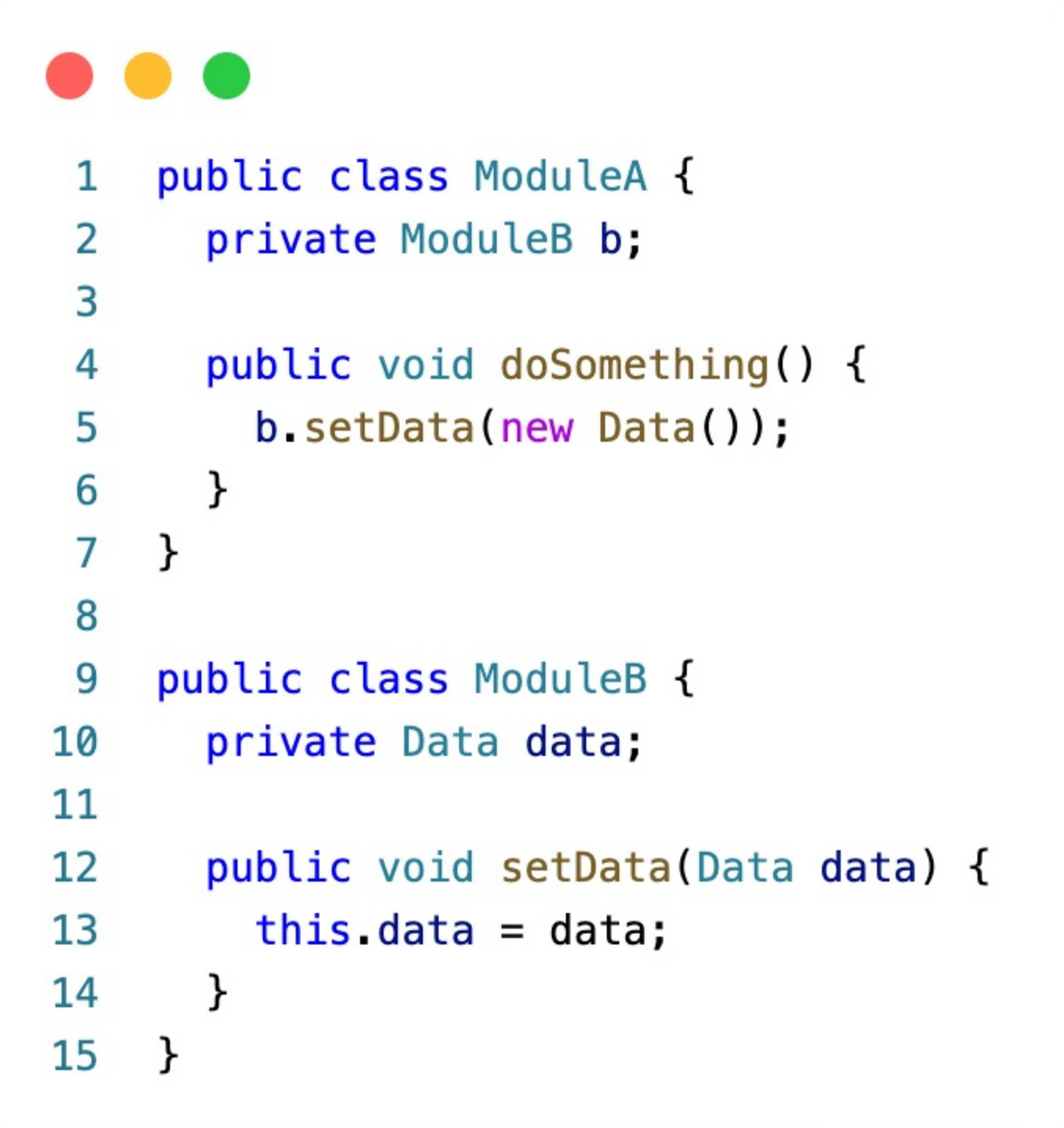


```java
public class OrderProcessor {
    public boolean processOrder(Order order) {
        // ... process the order
        boolean success = true; // or false if order processing failed
        EmailSender.sendEmail(success);
        return success;
    }
}

public class EmailSender {
    public static void sendEmail(boolean success) {
        if (success) {
            // ... send email
        } else {
            // ... send failure email
        }
    }
}
```

# Stamp Coupling

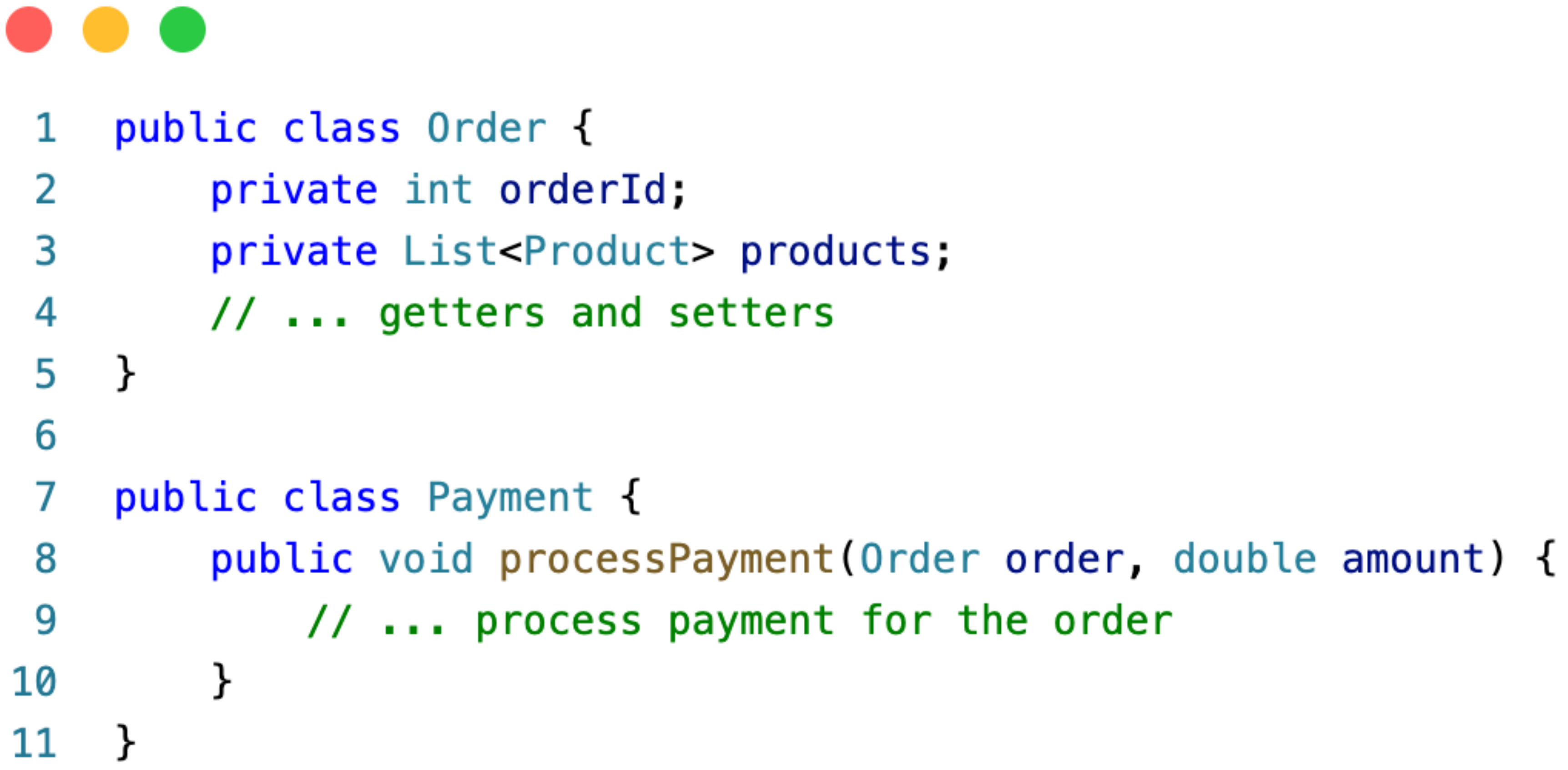
- This occurs when one module passes a large data structure to another module that only uses a small part of it.
  - Two or more modules share a composite data structure such as an array, struct or record, but only use part of it.
  - The modules only passing the whole data structure between them.
- This can lead to unnecessary dependencies and should be avoided when possible

# Example 1



```java
public class ModuleA {
  private ModuleB b;

  public void doSomething() {
    b.setData(new Data());
  }
}

public class ModuleB {
  private Data data;

  public void setData(Data data) {
    this.data = data;
  }
}
```

# Example 2

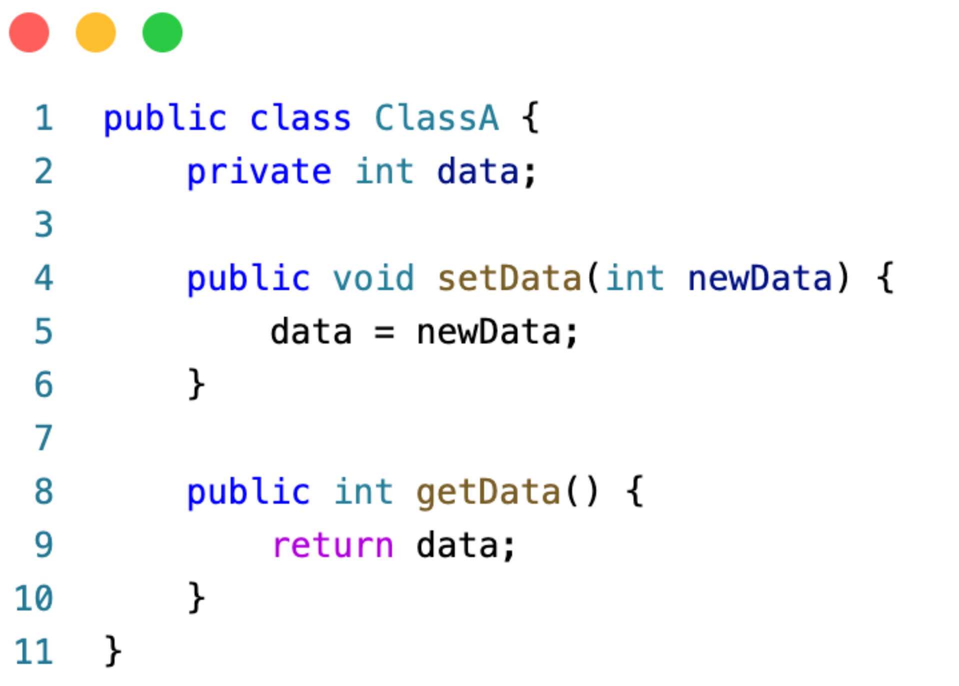


```java
public class Order {
    private int orderId;
    private List<Product> products;
    // ... getters and setters
}

public class Payment {
    public void processPayment(Order order, double amount) {
        // ... process payment for the order
    }
}
```

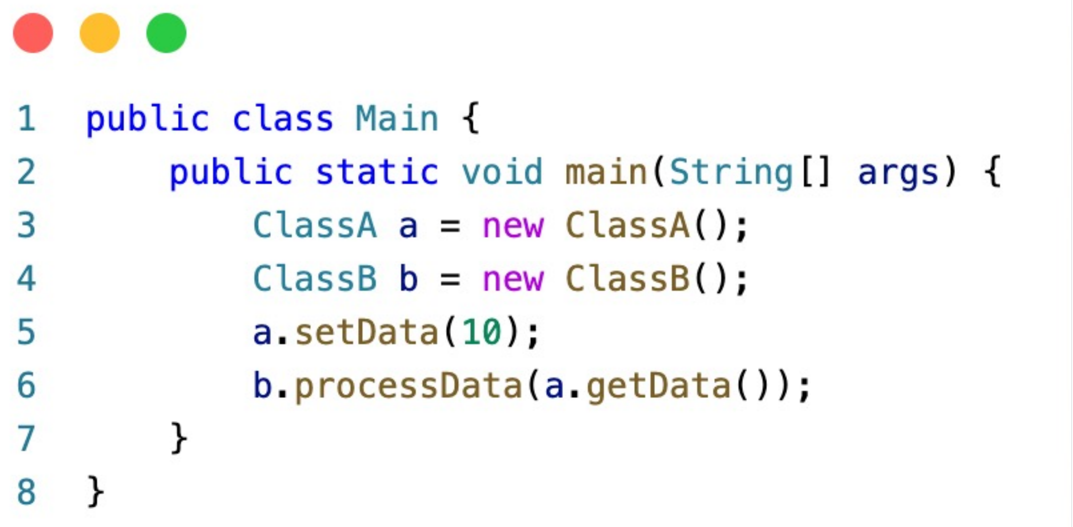
# Data Coupling

- Data coupling occurs when two modules share the same data.
- The modules do not communicate directly but share data through parameters and return values.
- This type of coupling is considered the weakest type of coupling.

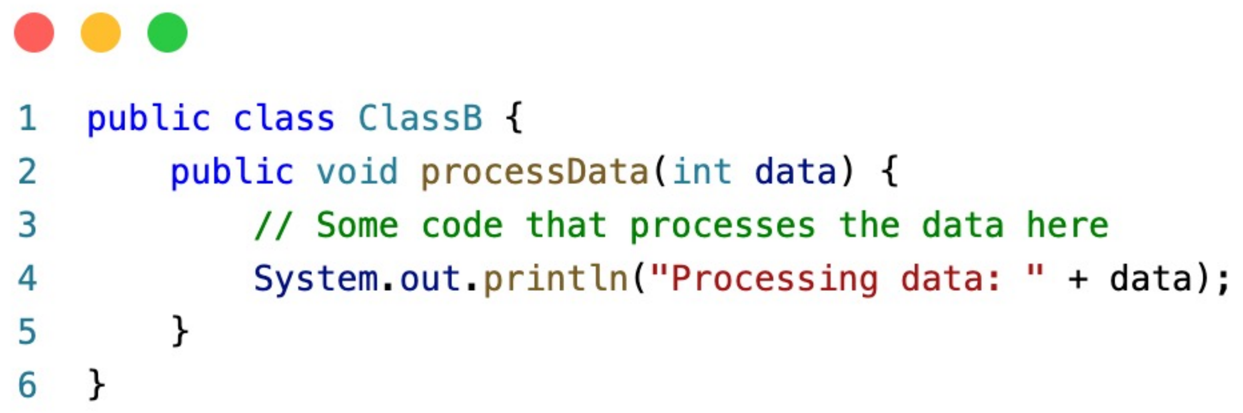
# Example



```java
public class ClassA {
    private int data;

    public void setData(int newData) {
        data = newData;
    }

    public int getData() {
        return data;
    }
}
```



```java
public class Main {
    public static void main(String[] args) {
        ClassA a = new ClassA();
        ClassB b = new ClassB();
        a.setData(10);
        b.processData(a.getData());
    }
}
```



```java
public class ClassB {
    public void processData(int data) {
        // Some code that processes the data here
        System.out.println("Processing data: " + data);
    }
}
```

# Modules Cohesion

# Introduction to Cohesion

- Module cohesion refers to the degree to which the elements within a module are related to each other and focused on achieving a single, well-defined purpose or responsibility.
- Modules with high cohesion are easier to understand, modify, maintain and can lead to increased reusability,

# Introduction to Cohesion cont.

- Modules with low cohesion can be more difficult to work with and understand because their components are scattered and unrelated.
- This can lead to code duplication, increased complexity, and a greater likelihood of errors and bugs.
- In extreme cases, modules with low cohesion may need to be completely rewritten or replaced, which can be time-consuming and costly.

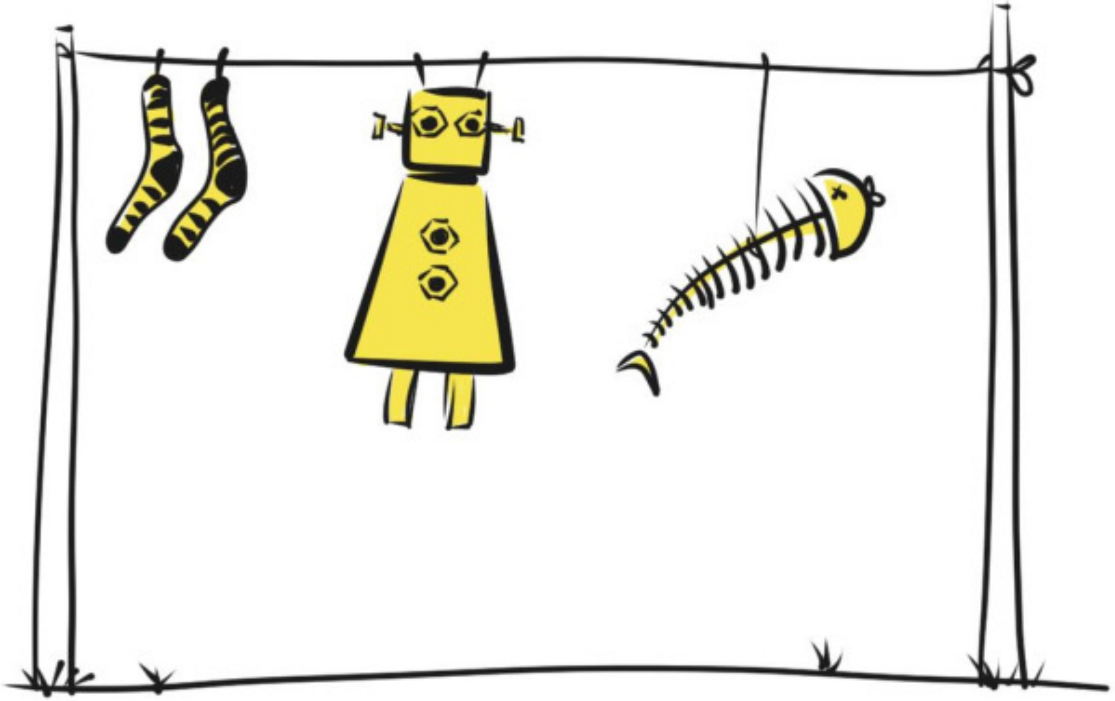
# Types of Cohesion

These types, in order of increasing cohesion, are:

- Coincidental cohesion
- Logical cohesion
- Temporal cohesion
- Procedural cohesion
- Communicational cohesion
- Sequential cohesion
- Functional cohesion

# Coincidental Cohesion

- The first criterion for assigning elements to a specific class could be an actual lack of any criteria.
- Total randomness in grouping elements into a class.
- Classes are created coincidentally and not in the process of design.

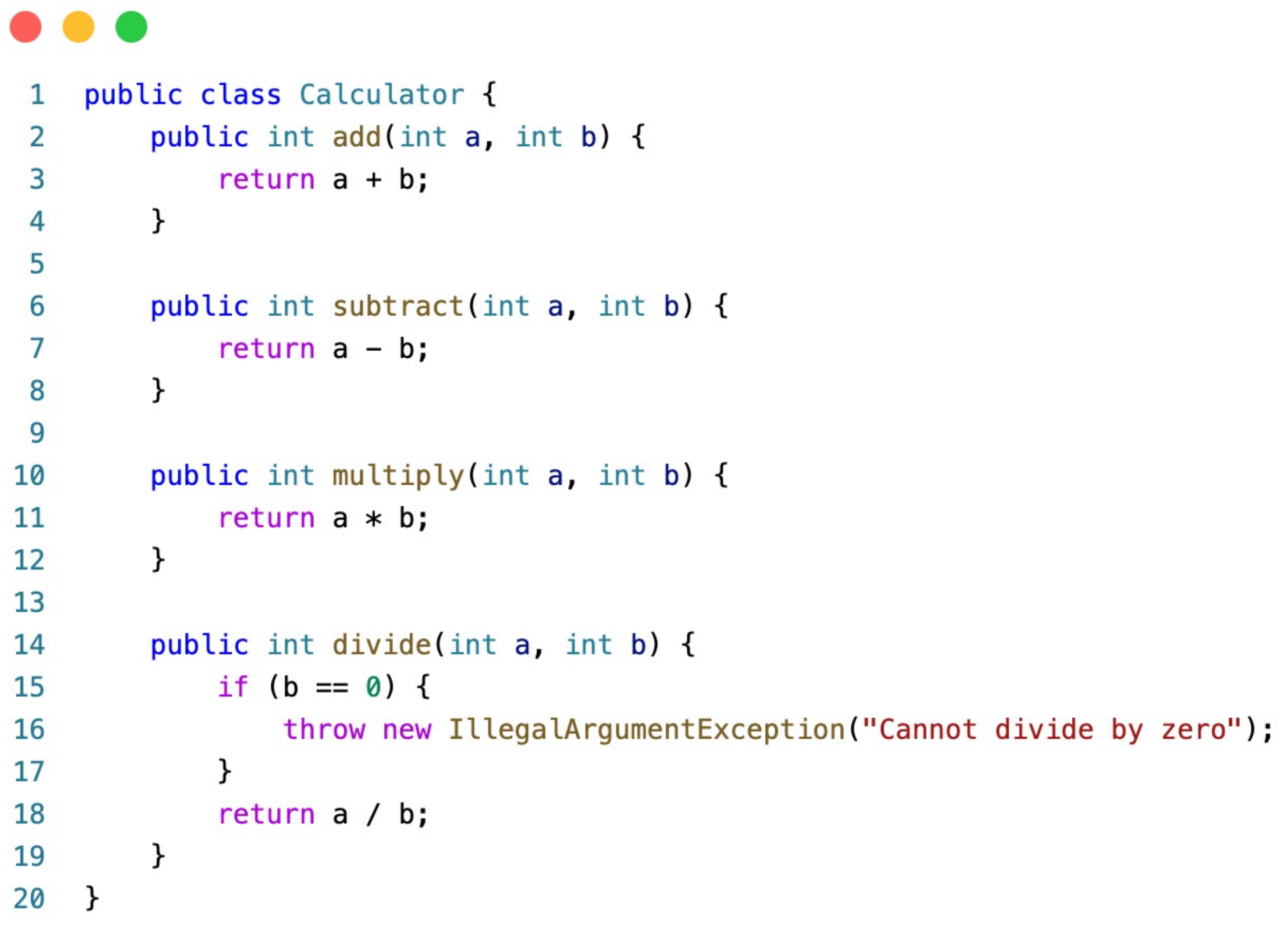


# Logical Cohesion

$$y = \left(\frac{2}{x}\right)^2 + \sqrt[3]{x}$$

- Elements of a class are grouped because they solve problems from the same category.
- Logically there is something that they have in common even though they have different functionalities.
- A good example here could be mathematical operations, as each of them can do various things, but often they are grouped in some Math or Utils class with… mathematical operations.

# Example

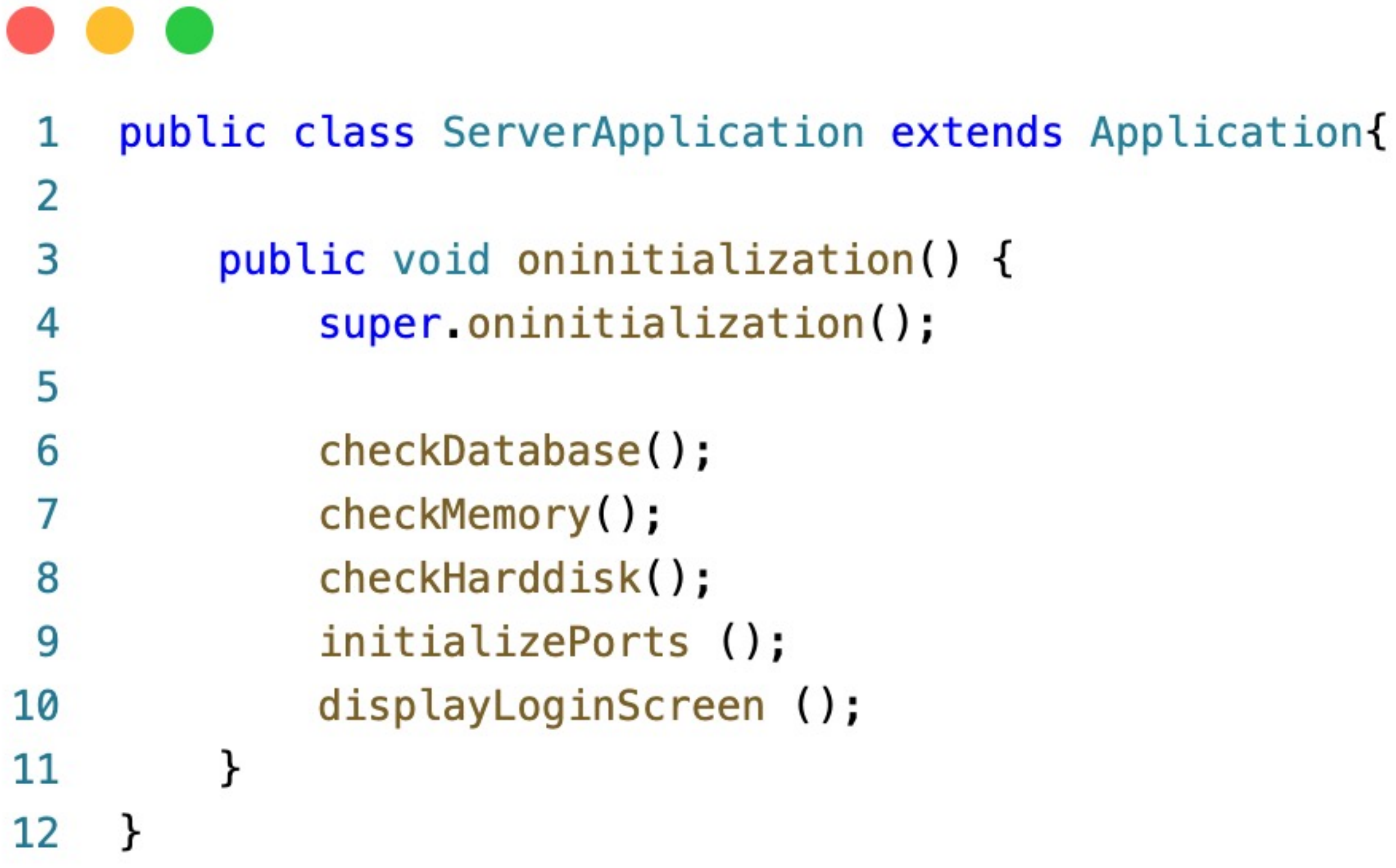


```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

# Temporal Cohesion

- Temporal cohesion – elements of the class have to be executed within the same period.

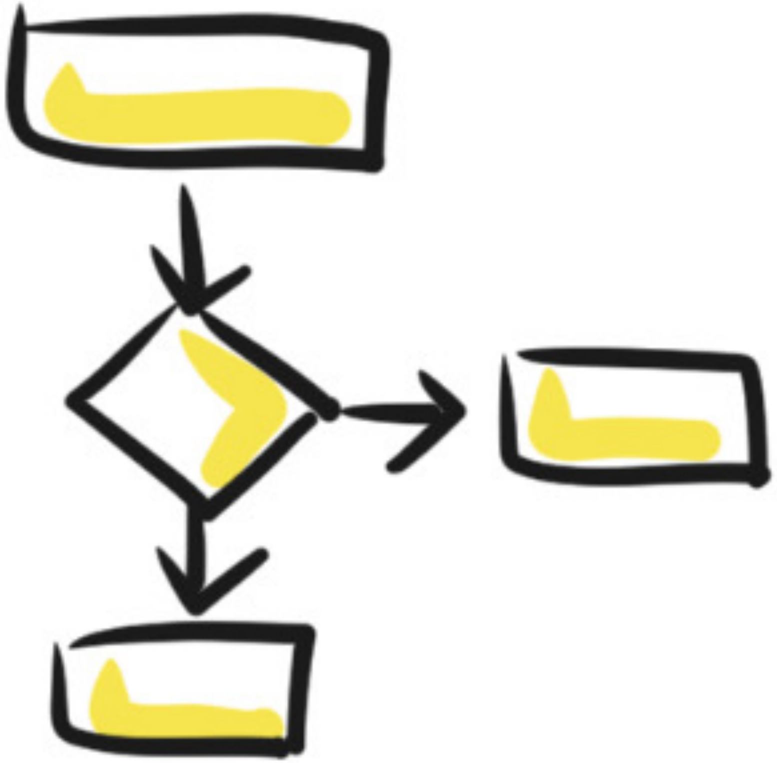
# Example



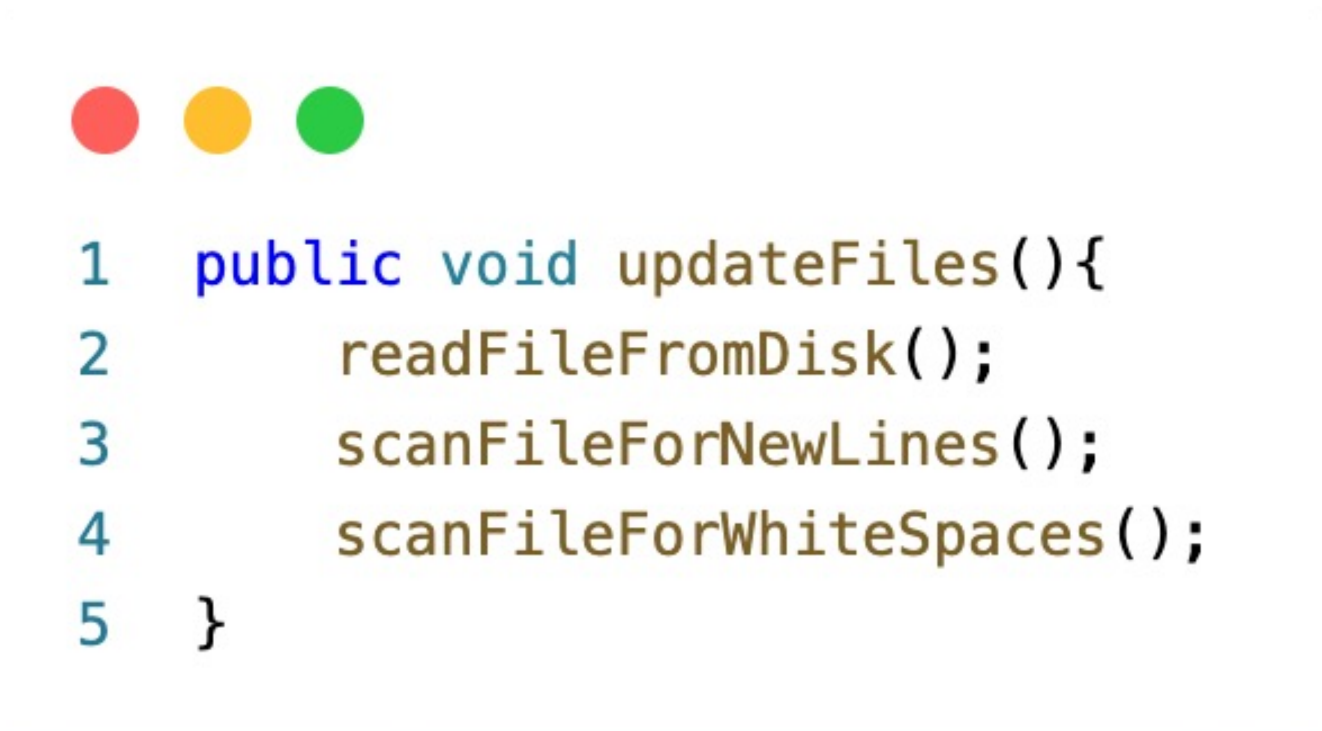
```java
public class ServerApplication extends Application{

    public void oninitialization() {
        super.oninitialization();

        checkDatabase();
        checkMemory();
        checkHarddisk();
        initializePorts ();
        displayLoginScreen ();
    }
}
```

# Procedural Cohesion

- The focus on an algorithm of execution – the order of steps that have to be executed to get from state “A” to state “B”.
- We model algorithm itself and not the problem that we were supposed to solve
- Loops, multiple conditions, and steps in the code can show evidence of procedural cohesion.



# Example

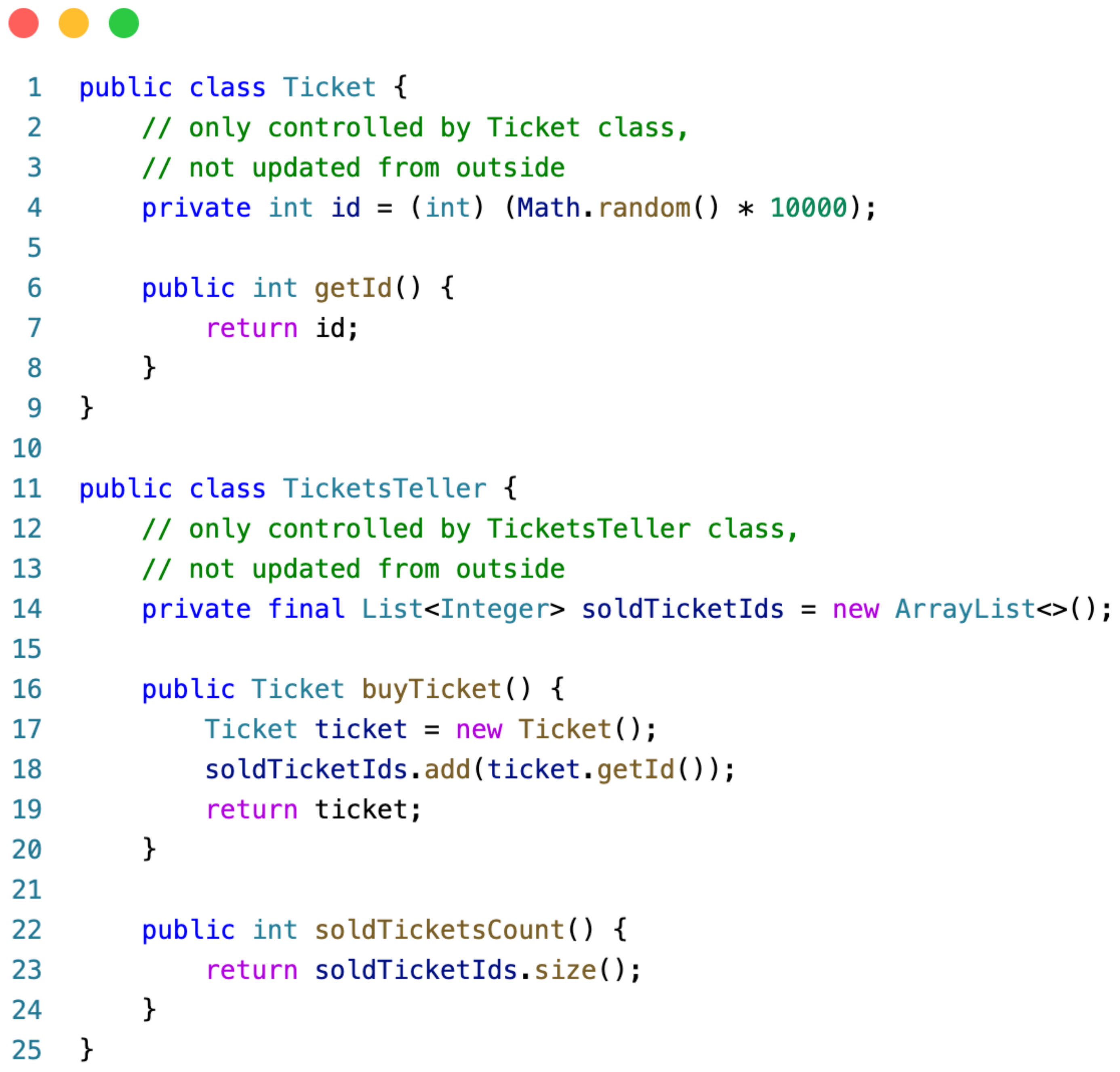


```java
public void updateFiles(){
      readFileFromDisk();
      scanFileForNewLines();
      scanFileForWhiteSpaces();
}
```

# Communicational Cohesion

- Communicational cohesion – elements in the class may perform different functions but are grouped because they are communicationally connected, so they use the same input data or return the same output data.
- This form of cohesion has clearly defined boundaries, inputs, and outputs.

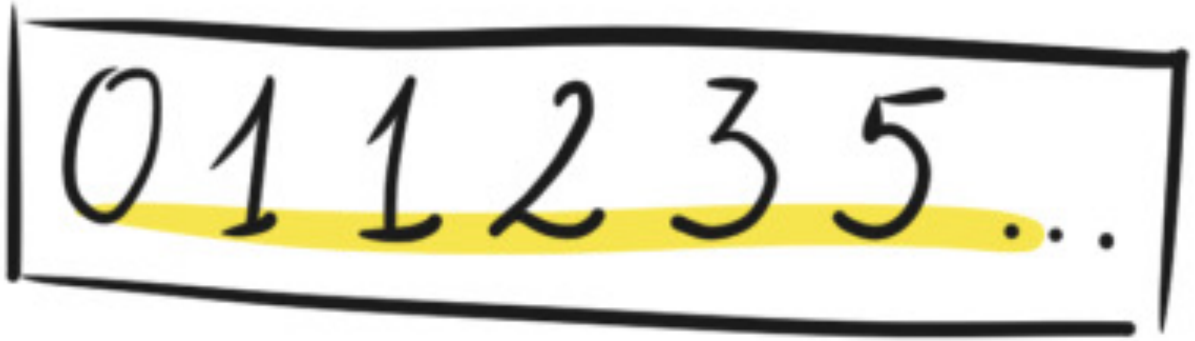
# Example



```java
public class Ticket {
    // only controlled by Ticket class,
    // not updated from outside
    private int id = (int) (Math.random() * 10000);

    public int getId() {
        return id;
    }
}

public class TicketsTeller {
    // only controlled by TicketsTeller class,
    // not updated from outside
    private final List<Integer> soldTicketIds = new ArrayList<>();

    public Ticket buyTicket() {
        Ticket ticket = new Ticket();
        soldTicketIds.add(ticket.getId());
        return ticket;
    }

    public int soldTicketsCount() {
        return soldTicketIds.size();
    }
}
```

# Sequential Cohesion

- At the stage when we group elements because of their sequence of data processing, so that each element relies on the output of the previous element.



# Example

Applying a series of filters to an image to enhance its appearance:

- Adjust the brightness and contrast of the image as needed.
- Apply a noise reduction filter to remove any unwanted artifacts.
- Apply a sharpening filter to enhance the edges and details
- Apply a color correction filter to adjust color balance and saturation.
- Apply a vignette filter to darken edges and draw focus to the center.
- Export the final image to a desired file format.

# Functional Cohesion

- This occurs when all elements within a class are there because they work together in the best possible way to accomplish the goal – functionality.
- Each element of such a class is its integral part and is critical for the functionality of the class
- The class itself performs no less and no more than one functionality.

# Examples

Examples of functional cohesive modules:

- Compute cosine of angle
- Read transaction record
- Assign seat to airline passenger