EMORY
UNIVERSITY

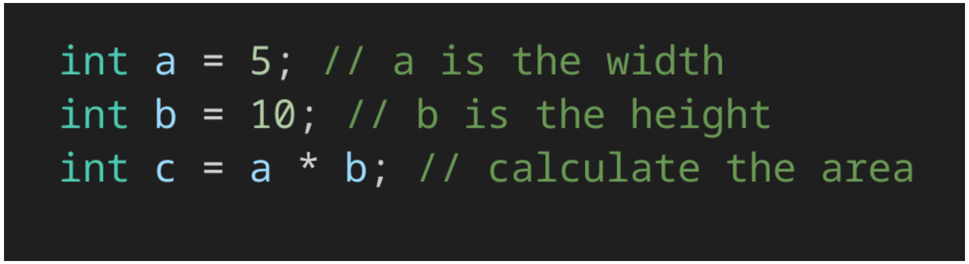
A Guide to Writing Efficient, Maintainable, and Elegant Code

Clean Code Development

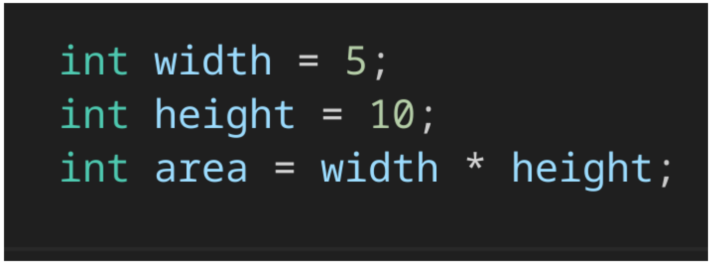
Hossein Amiri

Variables Naming

- Use names that describe the purpose of the variable



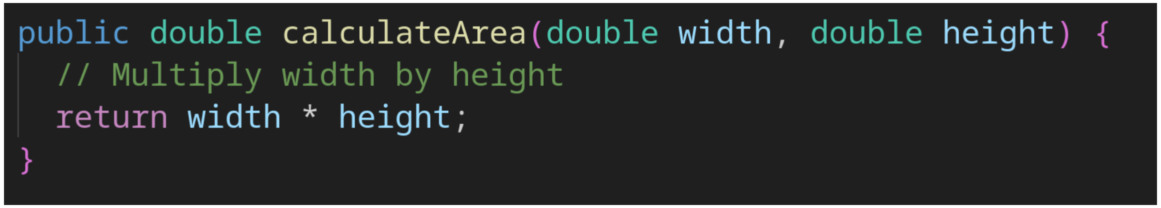
```
int a = 5; // a is the width
int b = 10; // b is the height
int c = a * b; // calculate the area
```



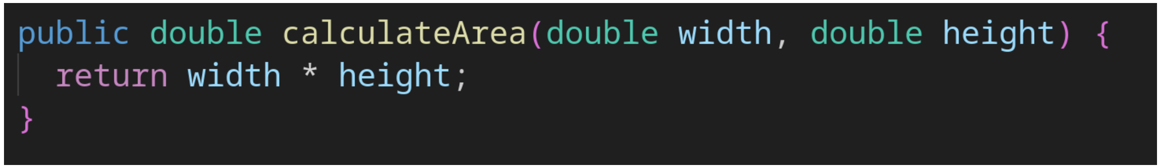
```
int width = 5;
int height = 10;
int area = width * height;
```

Comments

- Explain complex or non-obvious code
- Try to make the code self-explanatory

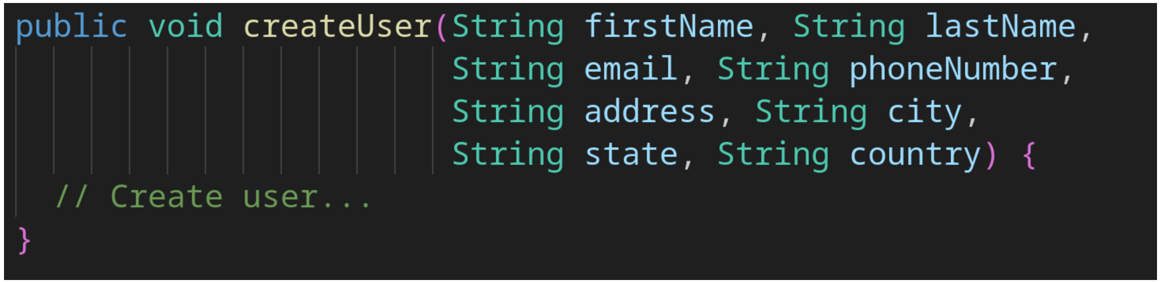


```java
public double calculateArea(double width, double height) {
  // Multiply width by height
  return width * height;
}
```

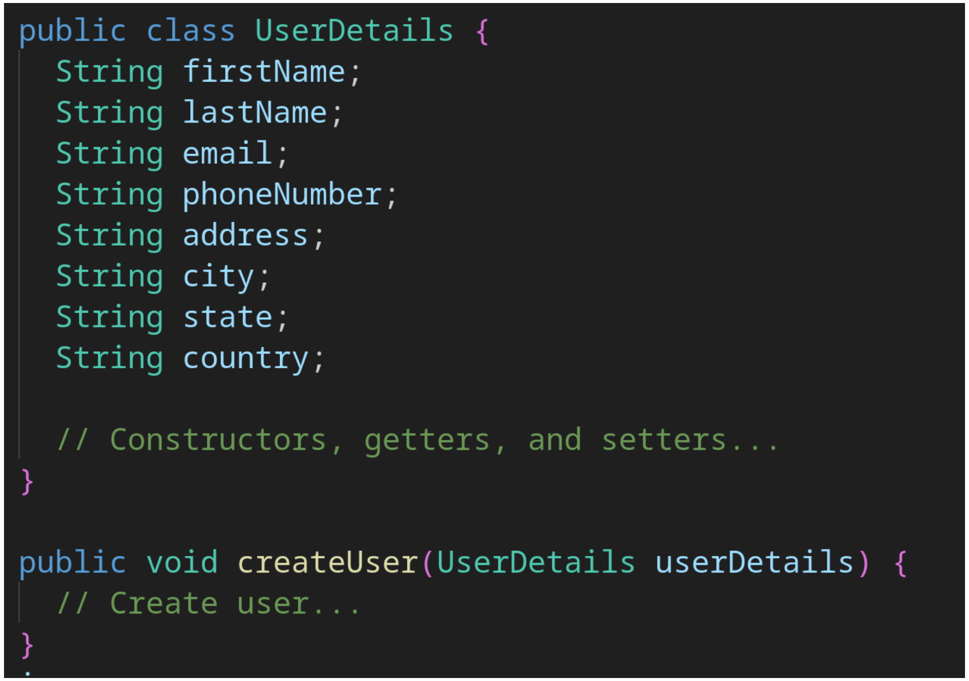


```java
public double calculateArea(double width, double height) {
  return width * height;
}
```

Long Parameter Lists

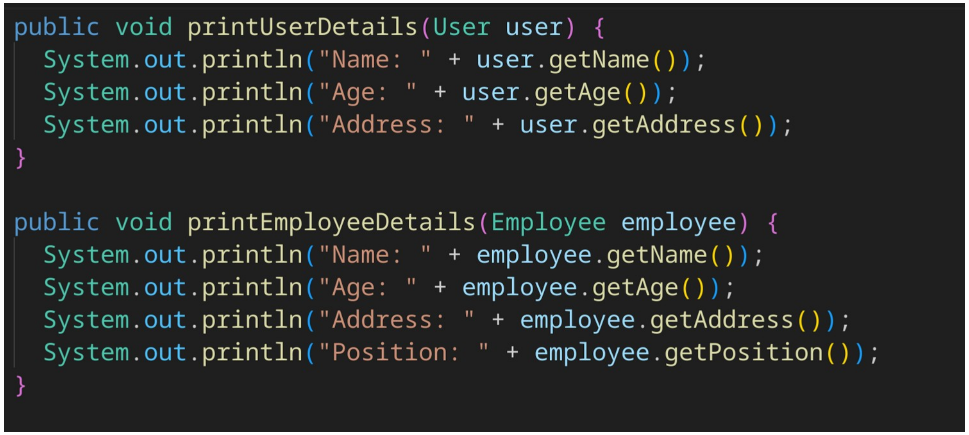


```java
public void createUser(String firstName, String lastName,
                       String email, String phoneNumber,
                       String address, String city,
                       String state, String country) {
  // Create user...
}
```

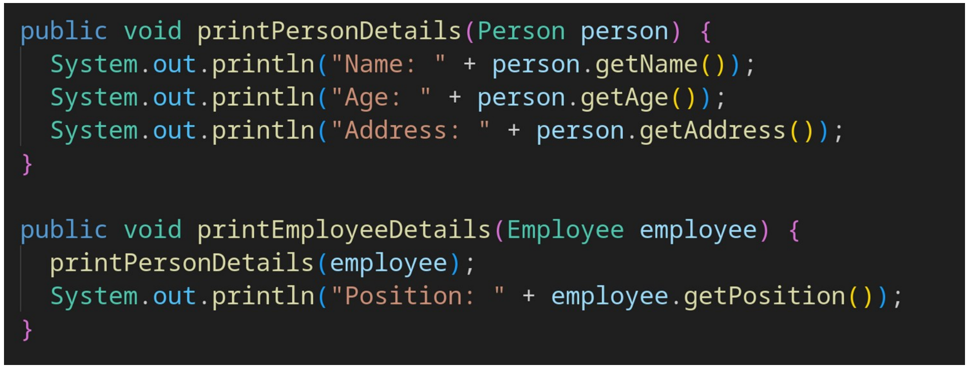


```java
public class UserDetails {
  String firstName;
  String lastName;
  String email;
  String phoneNumber;
  String address;
  String city;
  String state;
  String country;

  // Constructors, getters, and setters...
}

public void createUser(UserDetails userDetails) {
  // Create user...
}
```

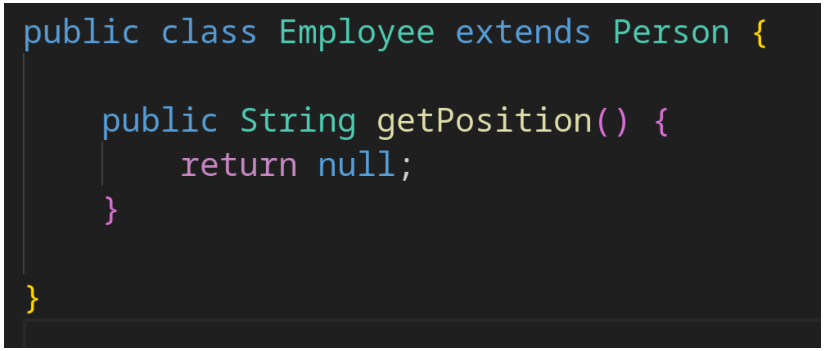
Code Duplication



```java
public void printUserDetails(User user) {
  System.out.println("Name: " + user.getName());
  System.out.println("Age: " + user.getAge());
  System.out.println("Address: " + user.getAddress());
}

public void printEmployeeDetails(Employee employee) {
  System.out.println("Name: " + employee.getName());
  System.out.println("Age: " + employee.getAge());
  System.out.println("Address: " + employee.getAddress());
  System.out.println("Position: " + employee.getPosition());
}
```



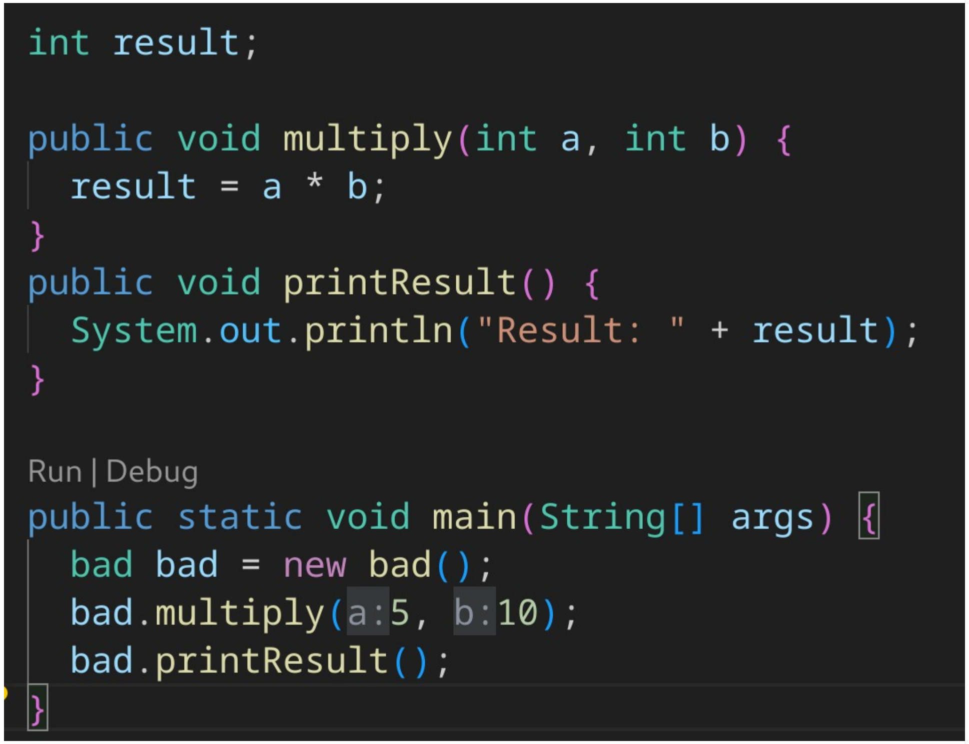
```java
public void printPersonDetails(Person person) {
  System.out.println("Name: " + person.getName());
  System.out.println("Age: " + person.getAge());
  System.out.println("Address: " + person.getAddress());
}

public void printEmployeeDetails(Employee employee) {
  printPersonDetails(employee);
  System.out.println("Position: " + employee.getPosition());
}
```



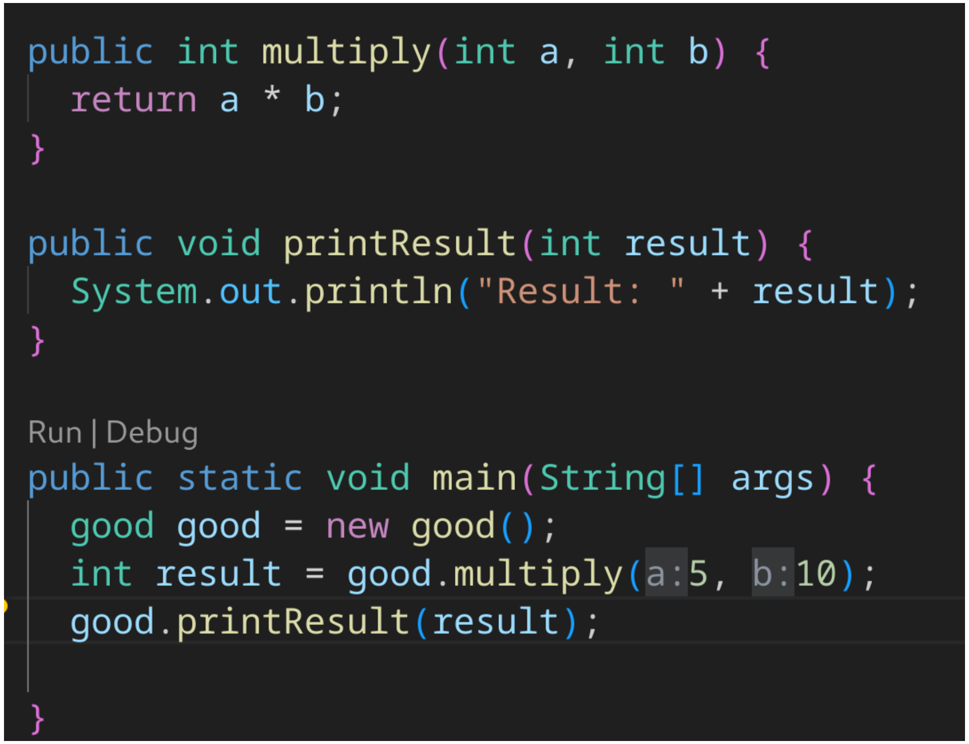
```java
public class Employee extends Person {

    public String getPosition() {
        return null;
    }

}
```

Variables Scope

- Limit the scope of variables to the smallest possible context



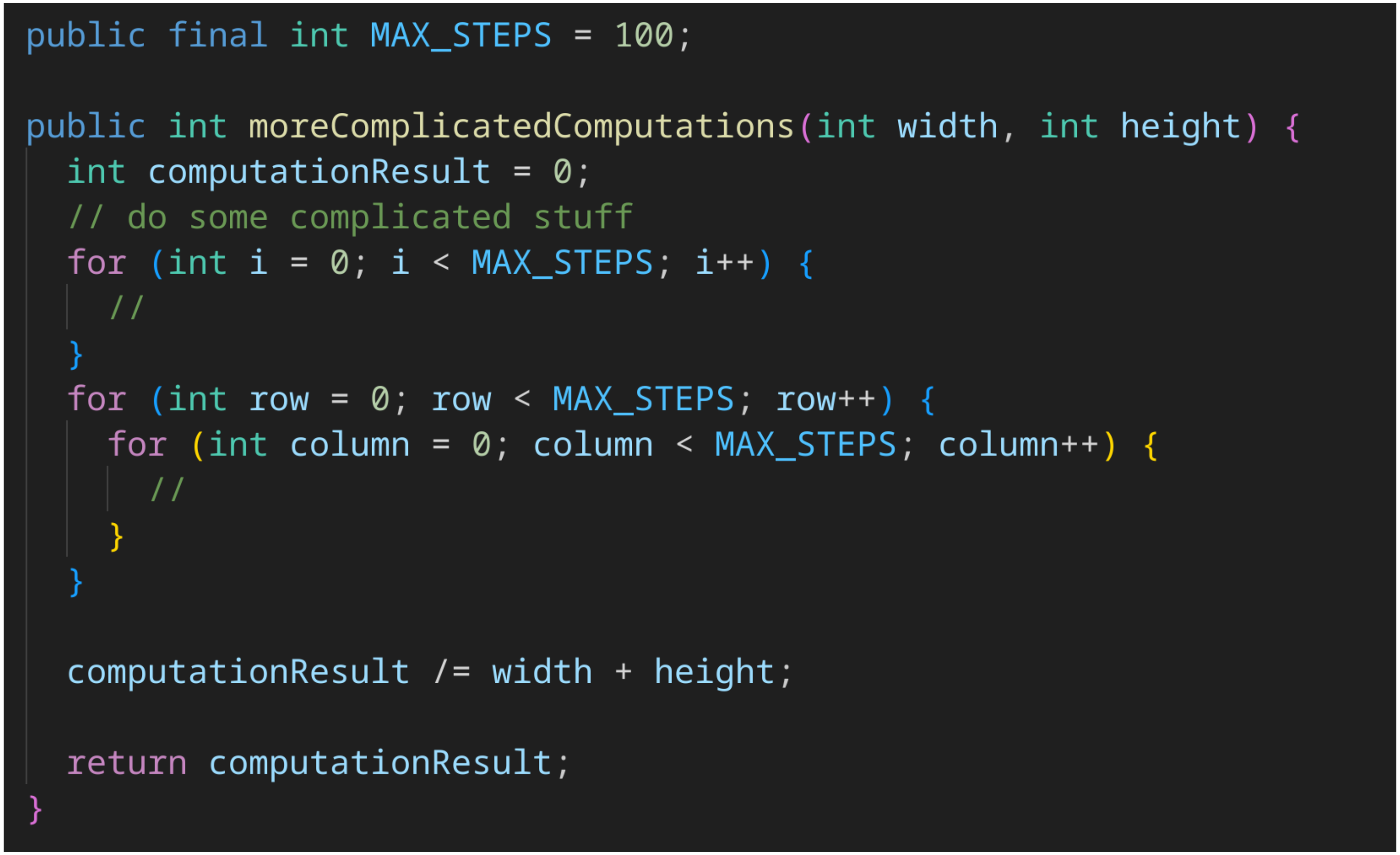
```java
int result;

public void multiply(int a, int b) {
  result = a * b;
}
public void printResult() {
  System.out.println("Result: " + result);
}

public static void main(String[] args) {
  bad bad = new bad();
  bad.multiply(a:5, b:10);
  bad.printResult();
}
```



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

public void printResult(int result) {
  System.out.println("Result: " + result);
}

public static void main(String[] args) {
  good good = new good();
  int result = good.multiply(a:5, b:10);
  good.printResult(result);

}
```

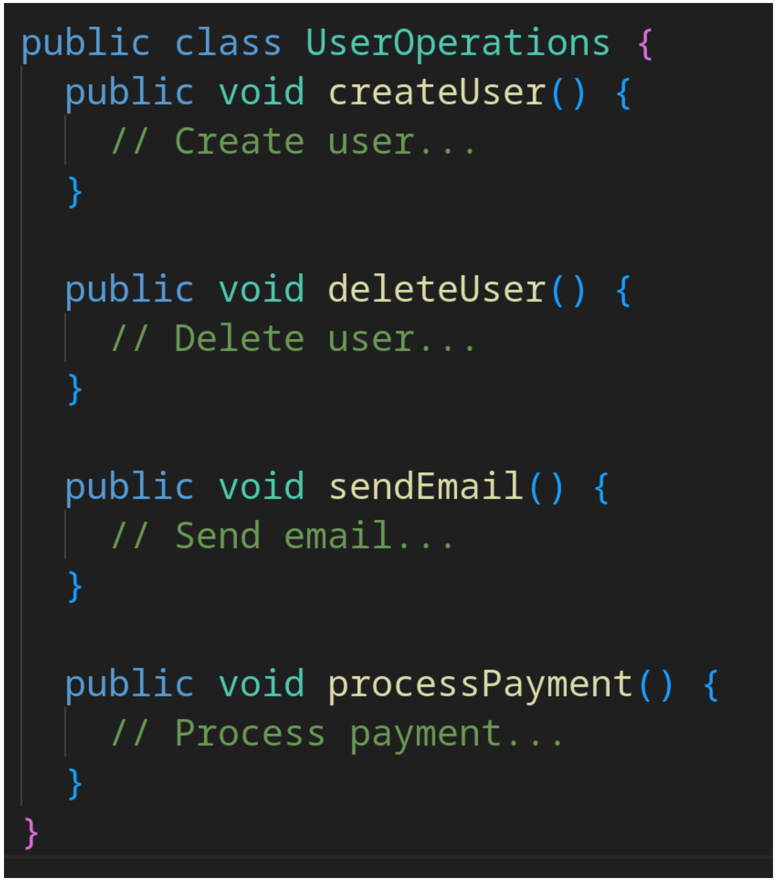
Variables Scope Naming

- The larger the scope is, the more comprehensive the variable name should be

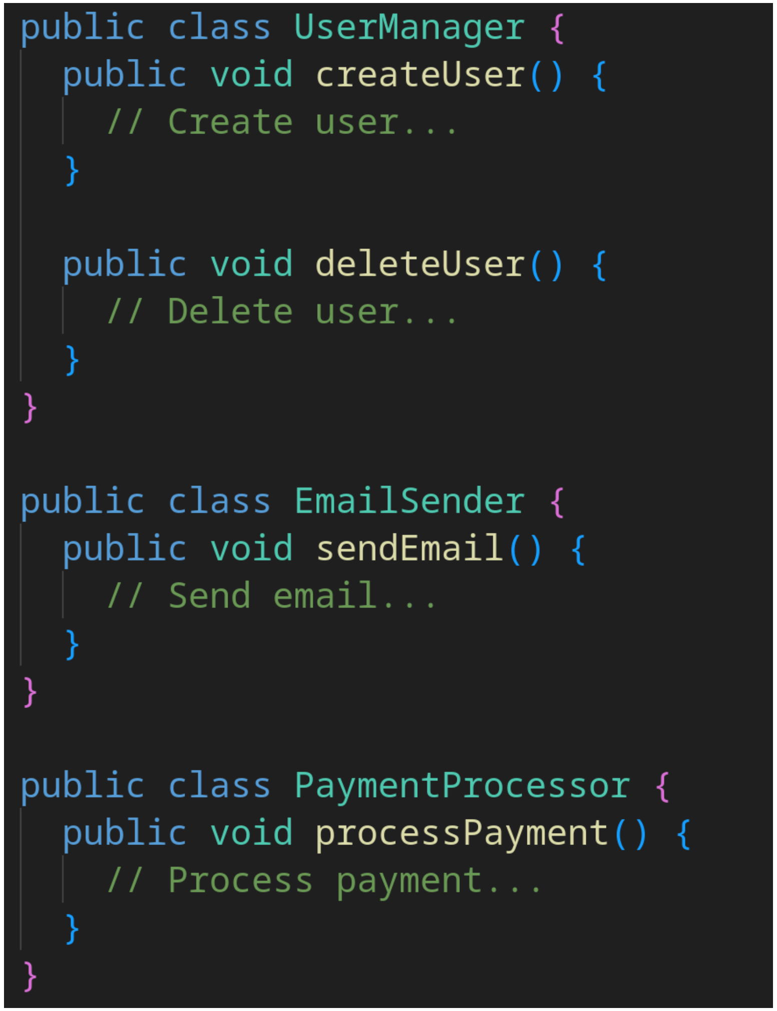


```java
public final int MAX_STEPS = 100;

public int moreComplicatedComputations(int width, int height) {
  int computationResult = 0;
  // do some complicated stuff
  for (int i = 0; i < MAX_STEPS; i++) {
    //
  }
  for (int row = 0; row < MAX_STEPS; row++) {
    for (int column = 0; column < MAX_STEPS; column++) {
      //
    }
  }

  computationResult /= width + height;

  return computationResult;
}
```

Code Organization



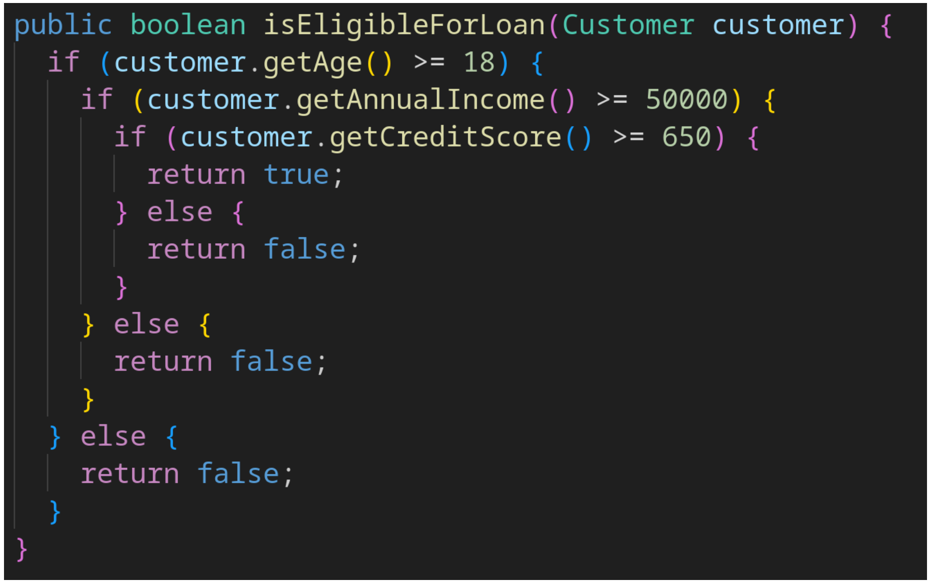
```java
public class UserOperations {
  public void createUser() {
    // Create user...
  }

  public void deleteUser() {
    // Delete user...
  }

  public void sendEmail() {
    // Send email...
  }

  public void processPayment() {
    // Process payment...
  }
}
```



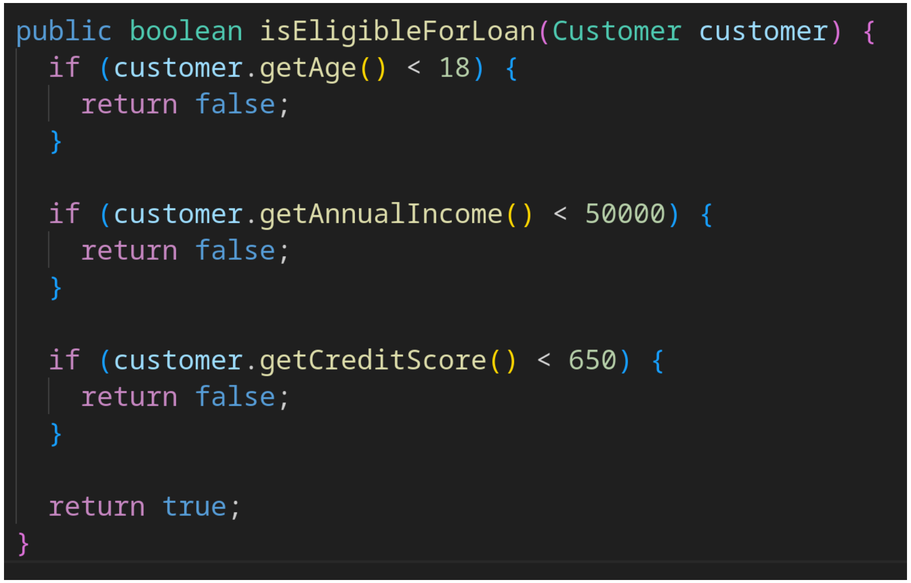
```java
public class UserManager {
  public void createUser() {
    // Create user...
  }

  public void deleteUser() {
    // Delete user...
  }
}

public class EmailSender {
  public void sendEmail() {
    // Send email...
  }
}

public class PaymentProcessor {
  public void processPayment() {
    // Process payment...
  }
}
```

Nested Conditionals

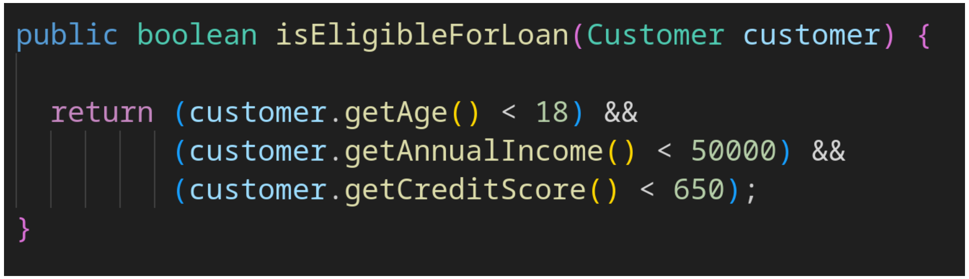
- Increases complexity and reduces readability



```java
public boolean isEligibleForLoan(Customer customer) {
  if (customer.getAge() >= 18) {
    if (customer.getAnnualIncome() >= 50000) {
      if (customer.getCreditScore() >= 650) {
        return true;
      } else {
        return false;
      }
    } else {
      return false;
    }
  } else {
    return false;
  }
}
```

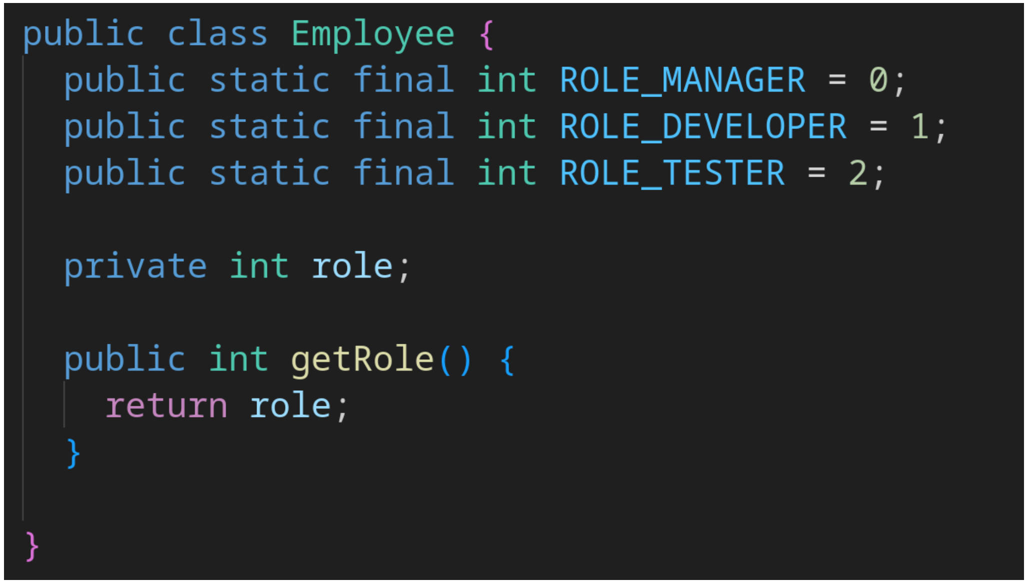


```java
public boolean isEligibleForLoan(Customer customer) {
  if (customer.getAge() < 18) {
    return false;
  }

  if (customer.getAnnualIncome() < 50000) {
    return false;
  }

  if (customer.getCreditScore() < 650) {
    return false;
  }

  return true;
}
```

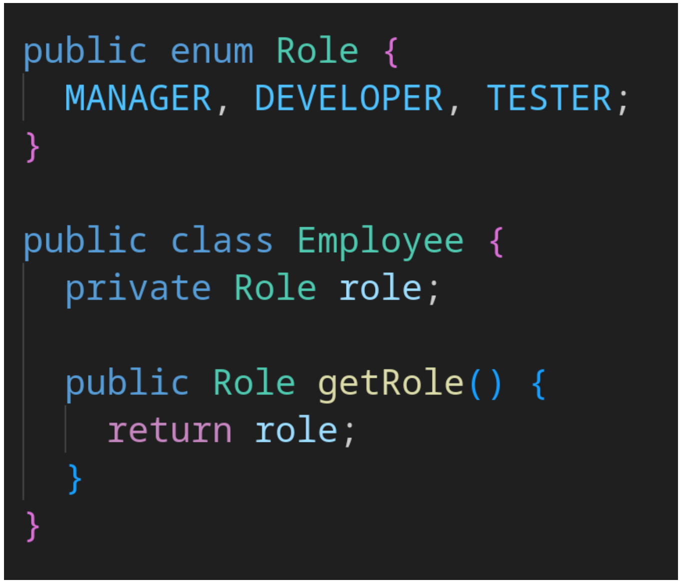


```java
public boolean isEligibleForLoan(Customer customer) {

  return (customer.getAge() < 18) &&
         (customer.getAnnualIncome() < 50000) &&
         (customer.getCreditScore() < 650);
}
```

Use Enums for Fixed Sets of Constants



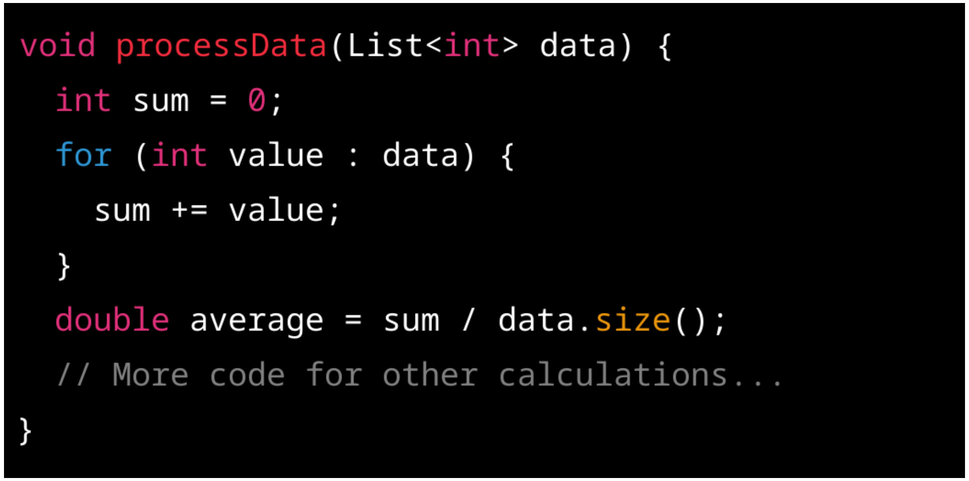
```java
public class Employee {
  public static final int ROLE_MANAGER = 0;
  public static final int ROLE_DEVELOPER = 1;
  public static final int ROLE_TESTER = 2;

  private int role;

  public int getRole() {
    return role;
  }

}
```



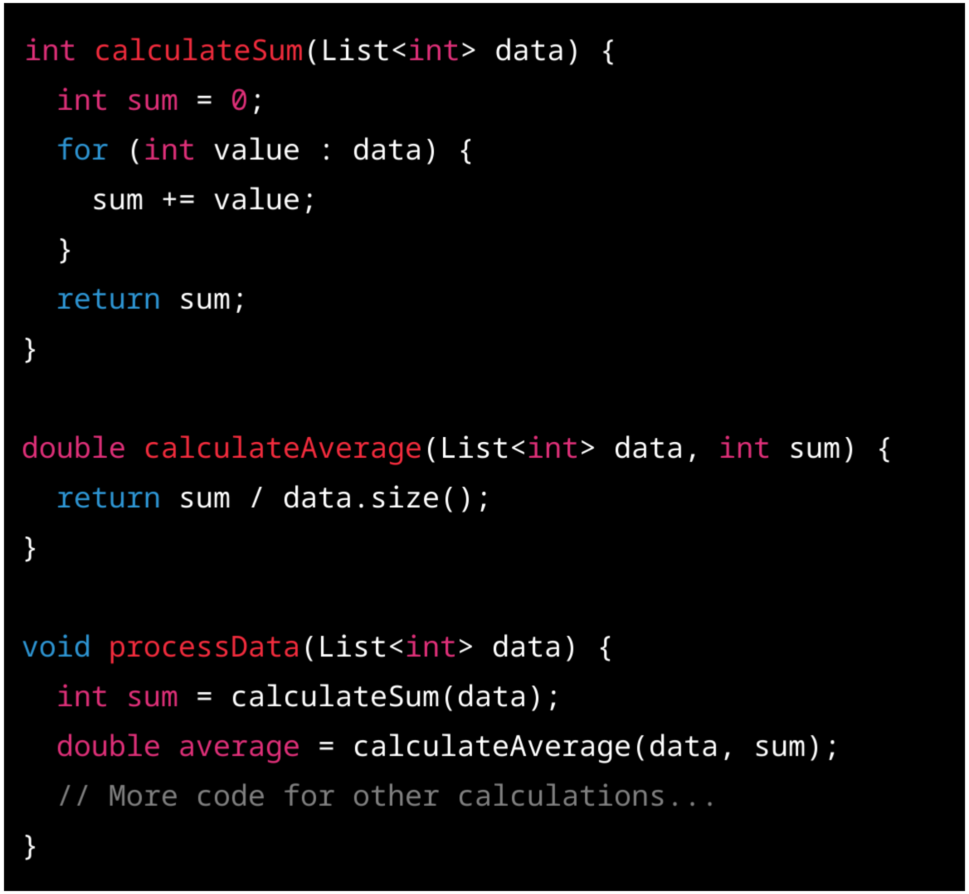
```java
public enum Role {
  MANAGER, DEVELOPER, TESTER;
}

public class Employee {
  private Role role;

  public Role getRole() {
    return role;
  }
}
```

Functions

- Keep functions small and focused



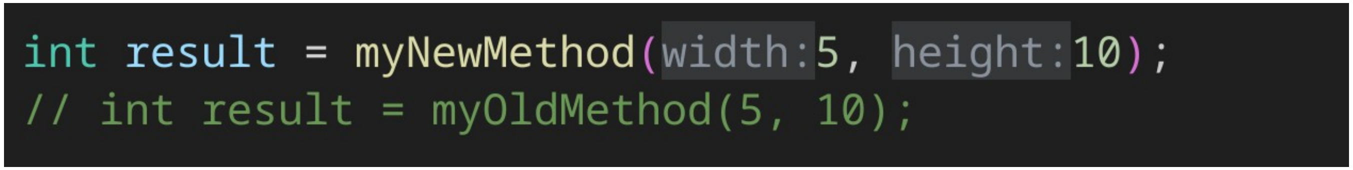
```
void processData(List<int> data) {
  int sum = 0;
  for (int value : data) {
    sum += value;
  }
  double average = sum / data.size();
  // More code for other calculations...
}
```



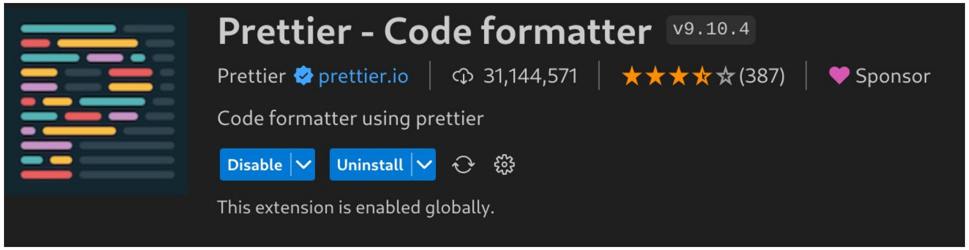
```
int calculateSum(List<int> data) {
  int sum = 0;
  for (int value : data) {
    sum += value;
  }
  return sum;
}

double calculateAverage(List<int> data, int sum) {
  return sum / data.size();
}

void processData(List<int> data) {
  int sum = calculateSum(data);
  double average = calculateAverage(data, sum);
  // More code for other calculations...
}
```

What Else?

- Error Handling
- Refactoring
- Dead Code
- Code Formatter



```
int result = myNewMethod(width:5, height:10);
// int result = myOldMethod(5, 10);
```



Prettier - Code formatter v9.10.4

Prettier prettier.io | 31,144,571 | ★★★★☆ (387) | Sponsor

Code formatter using prettier

This extension is enabled globally.