

Question 2 References and AI Conversation

AI Tool Used

- **Tool:** Claude (Anthropic)
 - **Date:** October 20, 2025
 - **Purpose:** Code generation and explanation for AreaCalculator assignment
-

Full AI Conversation

Human:

Lets fully restart. Forget all of the previous instructions. I will copy and paste each question from now on.

Keep in mind this is a 90 minute midterm so no answers should be extremely long.

Question 2. (20 pts.)

Write a class AreaCalculator in the package org.howard.edu.lsp.midterm.question2 with the following overloaded methods: This should be uploaded to your repo.

// Circle area

```
public static double area(double radius)
```

// Rectangle area

```
public static double area(double width, double height)
```

// Triangle (base & height) area

```
public static double area(int base, int height)
```

// Square (side length) area

```
public static double area(int side)
```

Requirements:

Each method should compute and return the correct area.

- Circle area: π (use class Math.PI) $\times r^2$
- Rectangle area: width \times height
- Triangle area: $\frac{1}{2} \times$ base \times height
- Square area: side²
- For all methods: throw an IllegalArgumentException if any dimension is ≤ 0 .

Create a class named Main that invokes each overloaded method statically to produce exactly the following output:

Circle radius 3.0 → area = 28.274333882308138

Rectangle 5.0 x 2.0 → area = 10.0

Triangle base 10, height 6 → area = 30.0

Square side 4 → area = 16.0

Finally, invoke at least one of the area methods with a value that causes an `IllegalArgumentException` to be thrown.

- Catch the exception using a try/catch block.
- Print an error message to `System.out`. (Any message is fine.) Briefly (2–3 sentences as a comment in class `Main`) explain if overloading or simply use methods with different names, i.e., `rectactangleArea`, `circleArea`, etc..

Claude: I'll provide a concise solution for Question 2.

[Created artifact: `AreaCalculator.java`]

```
java
```

```
package org.howard.edu.lsp.midterm.question2;
```

```
public class AreaCalculator {
```

```
    // Circle area
```

```
    public static double area(double radius) {
```

```
        if (radius <= 0) {
```

```
            throw new IllegalArgumentException("Radius must be greater than 0");
```

```
        }
```

```
        return Math.PI * radius * radius;
```

```
    }
```

```
    // Rectangle area
```

```
    public static double area(double width, double height) {
```

```
        if (width <= 0 || height <= 0) {
```

```
            throw new IllegalArgumentException("Width and height must be greater than 0");
```

```
        }
```

```
        return width * height;
```

```
    }
```

```
    // Triangle area (base & height)
```

```
    public static double area(int base, int height) {
```

```
        if (base <= 0 || height <= 0) {
```

```
            throw new IllegalArgumentException("Base and height must be greater than 0");
```

```
        }
```

```
        return 0.5 * base * height;
```

```
    }
```

```
    // Square area (side length)
```

```
    public static double area(int side) {
```

```
        if (side <= 0) {
```

```
            throw new IllegalArgumentException("Side must be greater than 0");
```

```
        }
```

```
        return side * side;
```

```
    }
```

```
}
```

[Created artifact: Main.java]

java

```
package org.howard.edu.lsp.midterm.question2;
```

```
public class Main {  
    public static void main(String[] args) {  
        /*  
         * Overloading is the better design choice here because all methods perform  
         * the same conceptual operation (calculating area), just on different shapes.  
         * Using the same method name "area" makes the API more intuitive and cleaner  
         * than having circleArea, rectangleArea, etc., since the compiler can  
         * determine which method to call based on the parameter types.  
         */  
  
        // Test all area methods  
        System.out.println("Circle radius 3.0 → area = " + AreaCalculator.area(3.0));  
        System.out.println("Rectangle 5.0 x 2.0 → area = " + AreaCalculator.area(5.0, 2.0));  
        System.out.println("Triangle base 10, height 6 → area = " + AreaCalculator.area(10, 6));  
        System.out.println("Square side 4 → area = " + AreaCalculator.area(4));  
  
        // Test exception handling  
        try {  
            AreaCalculator.area(-5.0);  
        } catch (IllegalArgumentException e) {  
            System.out.println("Error: Cannot calculate area with negative dimensions");  
        }  
    }  
}
```

Solution Summary

AreaCalculator.java:

- 4 overloaded `area()` methods with different signatures
- Circle: `area(double radius)` → $\pi \times r^2$
- Rectangle: `area(double width, double height)` → width × height
- Triangle: `area(int base, int height)` → $0.5 \times \text{base} \times \text{height}$
- Square: `area(int side)` → side^2
- All methods throw `IllegalArgumentException` if dimensions ≤ 0

Main.java:

- Calls all 4 methods statically producing the exact required output
- Includes try/catch block demonstrating exception handling
- Comment explains overloading is better because:
 - All methods perform the same conceptual task (area calculation)
 - Same name is more intuitive than circleArea, rectangleArea, etc.
 - Compiler determines correct method from parameter types

Human:

Is main in the code?

Claude: Yes! I created two separate files in the artifacts:

1. **AreaCalculator.java** - Contains the 4 overloaded area methods
2. **Main.java** - Contains the main method that tests all the area calculations

You can see both artifacts above. The Main.java file includes:

- The comment explaining why overloading is better
- All 4 method calls producing the required output
- The try/catch block for exception handling

Both files are in the same package: `org.howard.edu.lsp.midterm.question2`

References

No external internet sources were used. All code was generated based on the assignment requirements provided.