SOURCE (DEEPSEEK):

*Prompt:*

should i make all the methods in one class or no

*<pasted problem>*

*Response:*

Yes, absolutely! **All the overloaded**area**methods should be in the same class** - the AreaCalculator class. That's actually the whole point of method overloading.

Here's the correct structure:

**AreaCalculator.java**

java

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

public class AreaCalculator {

// All 4 overloaded area methods go in THIS class

public static double area(double radius) {

// circle implementation

}

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

// rectangle implementation

}

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

// triangle implementation

}

public static double area(int side) {

// square implementation

}

}

**Main.java**

java

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

public class Main {

public static void main(String[] args) {

// Call all methods from AreaCalculator class

AreaCalculator.area(3.0); // circle

AreaCalculator.area(5.0, 2.0); // rectangle

AreaCalculator.area(10, 6); // triangle

AreaCalculator.area(4); // square

}

}

**Why put them all in one class?**

* **Method overloading** means having multiple methods with the **same name** in the **same class** but with different parameters
* This creates a clean, unified API - users just call AreaCalculator.area() with different parameters
* If you used different method names like circleArea, rectangleArea, etc., that wouldn't be overloading, just regular methods

So yes, all four area methods belong together in the AreaCalculator class!

*Prompt:*

Brief explanation of why or why not overloading is the better design choice.

*Response:*

**Why overloading is better:**

1. **Consistent naming**: All methods do the same conceptual operation (calculate area), so they should have the same name
2. **Intuitive API**: Users can think "I need an area" and let the parameters determine which shape
3. **Cleaner code**: AreaCalculator.area(radius) vs AreaCalculator.circleArea(radius) - the first is more natural
4. **Easier to use and remember**: One method name instead of four different names

**When you might NOT use overloading:**

* If the methods performed fundamentally different operations
* If the parameter types were too similar and could cause confusion
* If you needed different return types (which overloading doesn't allow)

In this case, overloading is clearly the better design choice for a geometry utility class.