Software Engineering, Abstraction and Other Concepts

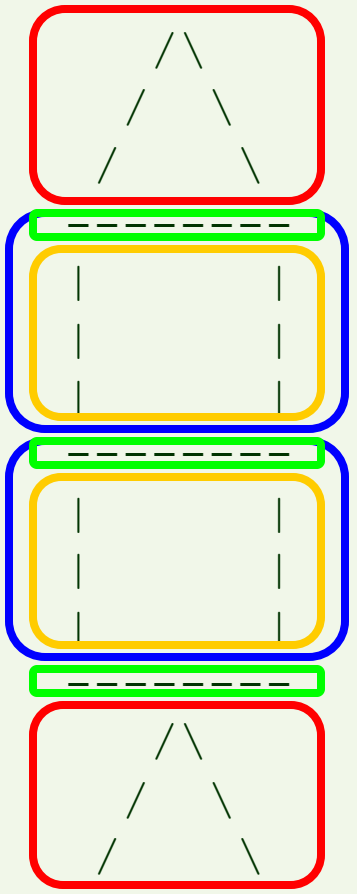
Learning Outcomes

- Draw a structure chart for a given program
- Write methods with good control abstraction, tight cohesion and loose coupling
- Identify the three types of scope, and where and how to use them
- Design effective interactive input/output

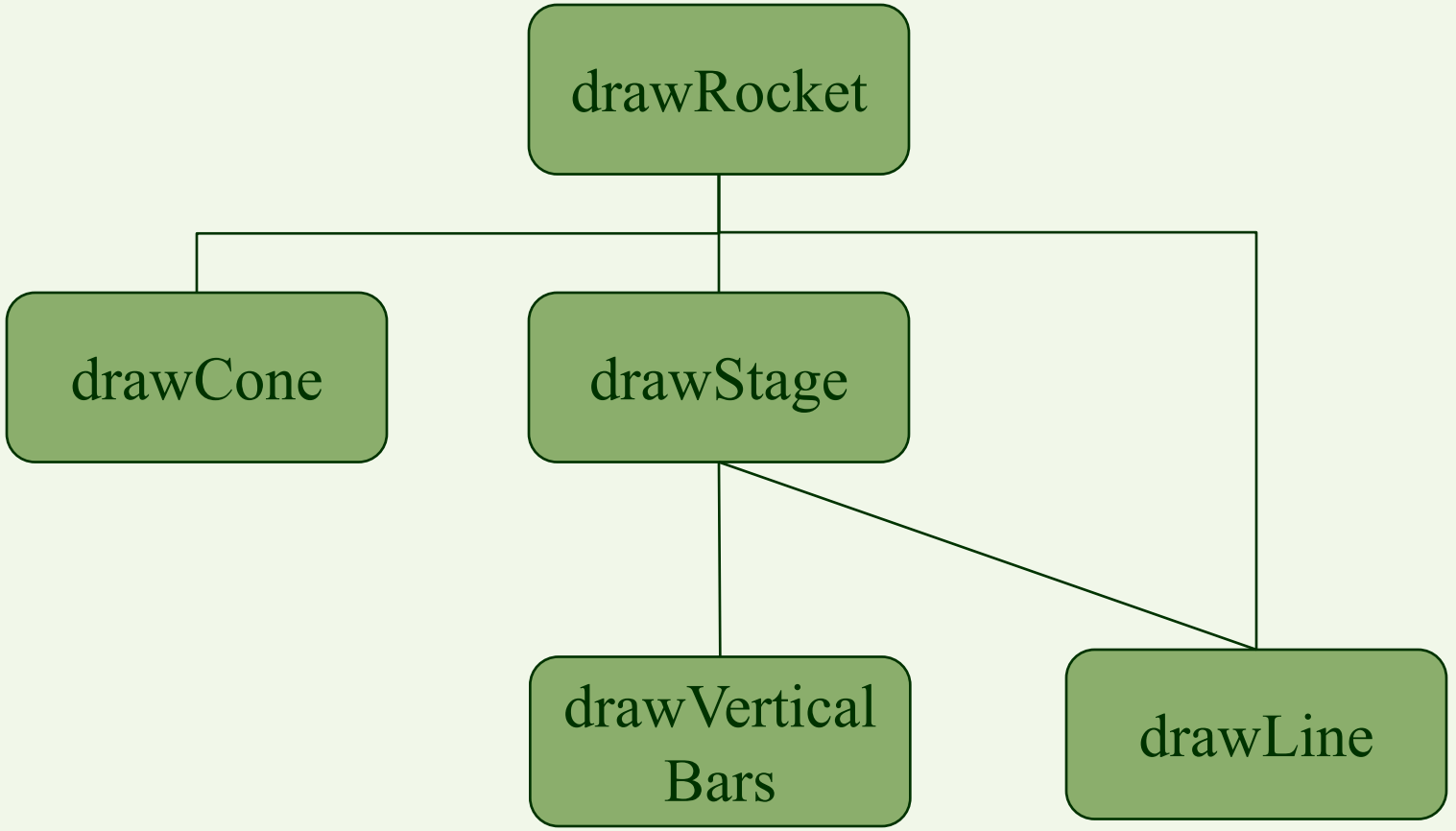
Rocket

Algorithm DrawRocket

1. Call DrawCone
2. Call DrawStage
3. Call DrawStage
4. Call DrawLine
5. Call DrawCone
6. Stop



Structure Chart



Abstraction

- Definition
  - A moving away from the details to the concepts

    - Control Abstraction (procedural)
    - Data Abstraction

Control Abstraction

- A separation of the logical properties of an action from its implementation

```
private static void drawRocket()
{
        drawCone();
        drawStage();
        drawStage();
        drawLine();
        drawCone();
}
```

WHAT
is more
important than
HOW

Functional Cohesion

Principle

- A module should perform exactly one abstract action
  - If you can describe the actions of a module in one simple sentence – it's probably cohesive
  - If you need to use the word "and" – it's doing too much

Communication Complexity (Coupling)

Principle

- Measure of the quantity of data that passes through the module's interface
  - It's better to have less parameters and no class or global variables
  - … a need to know basis

Rules of Thumb

- Use good abstraction
  - What, not how
- Have strong cohesion
  - Does one job
- Have loose coupling
  - Less parameters, no globals

Scope

*Region of a program where it is **legal** to reference/use an identifier.*

- Local Scope
  - Region extends from the declaration to the end of the block
  - Declared within a method
- Class Scope
  - Declared within a class but outside the methods
- Global Scope
  - Declared outside all classes and methods

Scope Example

```
import java.util.Scanner;

public class Menu
{
    final private static char EXIT = 'E';
    private static int c3;

    public static void main (String[] args)
    {
        go();
    }

    private static void go ()
    {
        int c3;
        c3 = 27;
    }
```

```
    private static void goAgain(int args)
    {
        int b2;
        while ( ___ )
        {
            int c1;
            :
            :
            args = ...
            b2 = ...
            :
            :
        }
        c1 = ...
        b2 = ...
    }

    :
}
```

Java Scope Rules

- Only the predefined packages are in the global scope
- All user defined method names are in the class scope
- Formal parameters are local to that method only
- Class variables / constants are in the class scope
- Local variables / constants extend from their declaration to the end of the block
- If a local identifier is declared with the same name as a class identifier, then the local identifier takes precedence

Interactive Input/Output

- Give titles, instructions, and purpose of the program

```
System.out.println(
    “Shape Drawing Program\n” +
    “The program will draw any ” +
    “shape selected from the menu.” );
```

Interactive Input/Output

- Prompt user for input values

```
System.out.print( “Please enter the first letter of ” +
                                “the shape to draw:  ” );
command = keyboard.next();
```

Interactive Input/Output

- If appropriate, give a menu of valid choices and how to exit the program

```
System.out.println( "Shape Menu" );
System.out.println( "==========" );
System.out.println( "(E)xit" );
System.out.println( "(T)riangle" );
```

Interactive Input/Output

- If an input is invalid (or out of range) then say so. Tell what are valid values and give a second chance.
  - It's often a good idea to echo the erroneous input

```
System.out.println( "Sorry but \"" + answer +
                    "\" is not a valid command.");
System.out.println( "Valid commands are E and T");
```

- Label and describe all your output