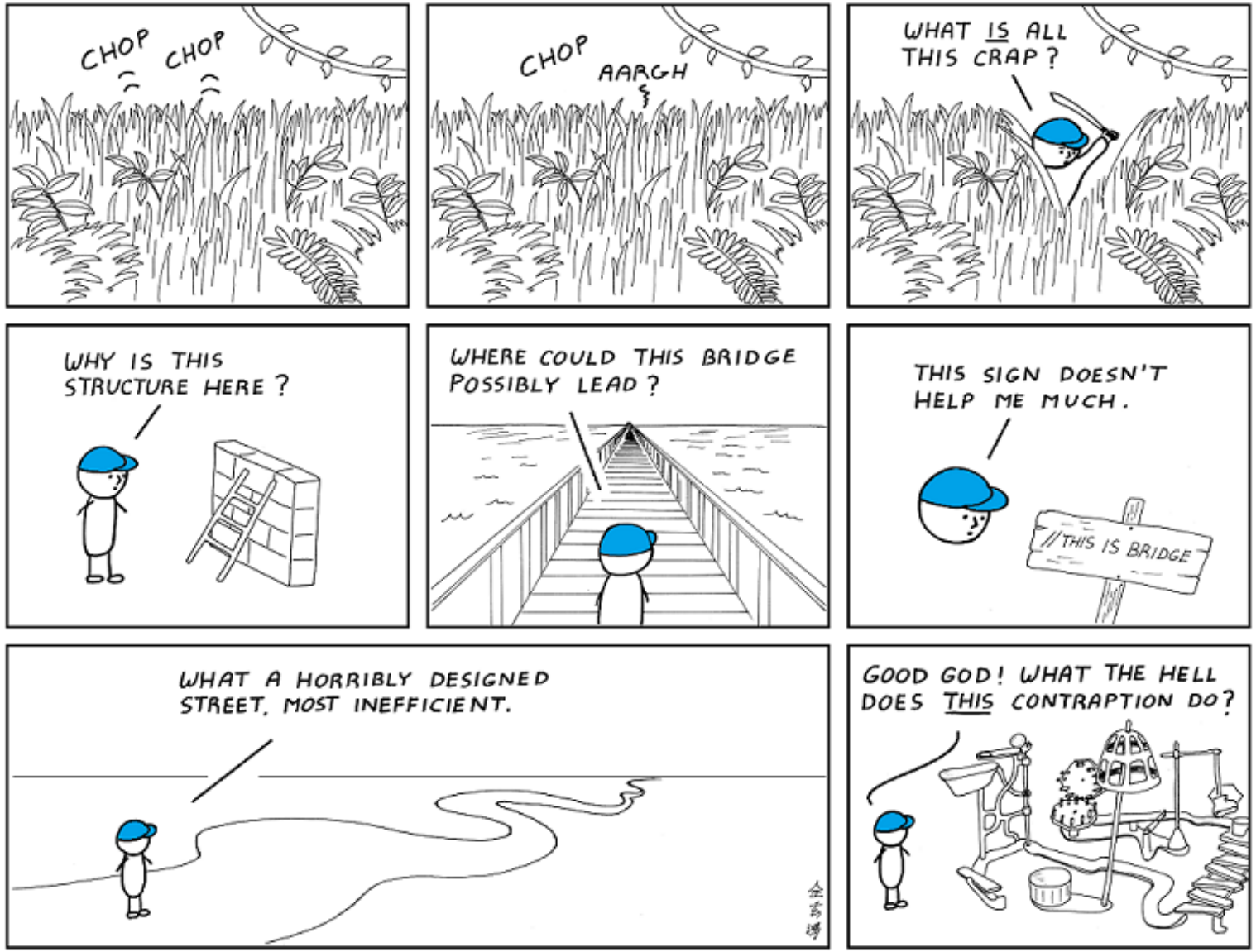


I hate reading other people's code.

Code Comprehension

CSCI 2134: Software Development

Agenda

- Code comprehension (precursor to refactoring)
- Brightspace Quiz
- Readings:
  - This Lecture: Chapter 8
  - Next Lecture: Chapter 24

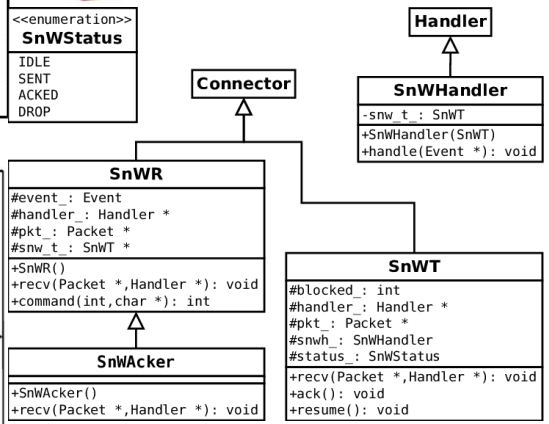
Program Comprehension

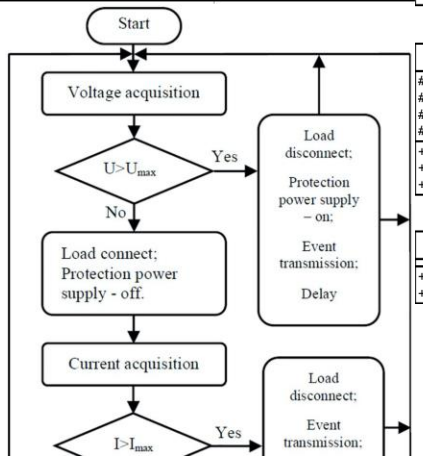
- The task of understanding someone else's code
- Why is this hard?
  - There is lots of code (many classes, methods, data structures, etc…)
  - Typically, no road map (no big picture)
  - Different style guide may have been used
  - Many things may be done differently than how "you" would do them
  - Not clear where to start ☹
- Why do we need to do this?
  - Need to extend other people's code
  - Need to debug other people's code
  - Need to refactor other people's code
  - Because my boss told me to

What Do We Need to Comprehend

- What is the problem being solved?
- What are the software requirements?
- What are the classes and their purpose?
- How do the classes collaborate?
  What design patterns are being used?
- How is each class implemented?
  - What data structures does it use?
  - What are the relationships between classes?
    - Inheritance
    - Aggregation
    - Nesting
- What is the control flow through the software?
- What is the flow of data through the software?

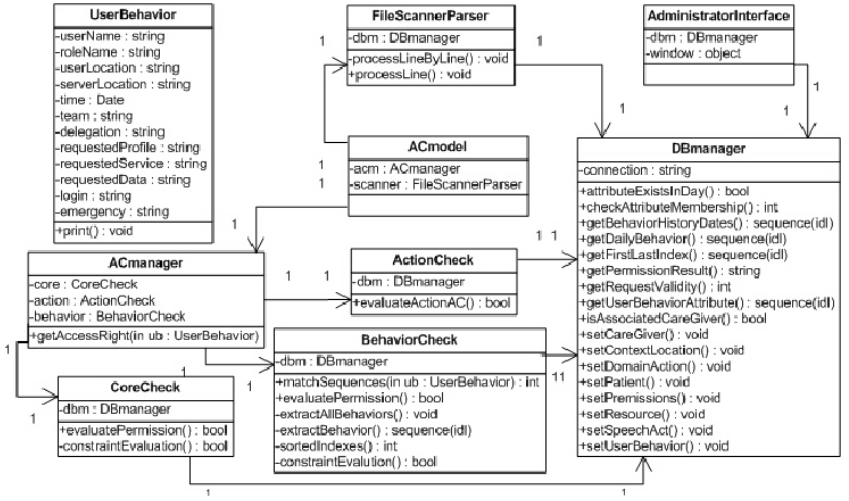
Class CardReader	
Responsibilities	Collaborators
Tell ATM when card is inserted	ATM
Read information from card	Card
Eject card	
Retain card	

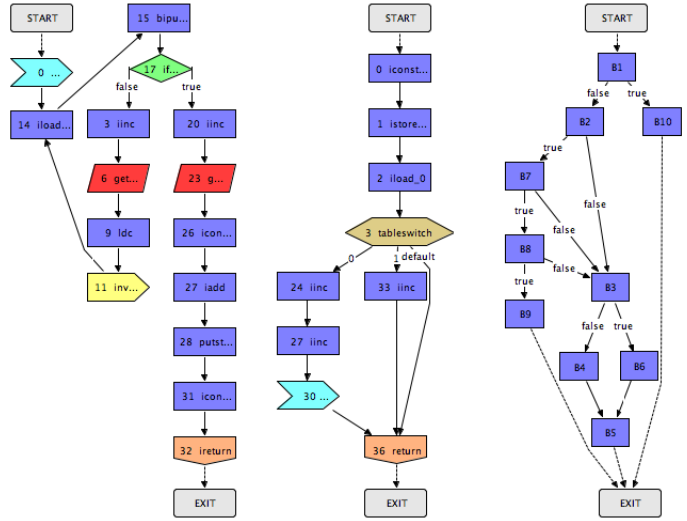




Three General Approaches

- Step 0: Understand the problem
- Step 1: Choose an approach
  - Top-Down: Class relationships and collaborations ☺
  - Control-Flow: What happens next? ☺
  - Bottom-Up: Understand the small components work-up
- Step 2: Begin exploration
- Step ∞: Finish



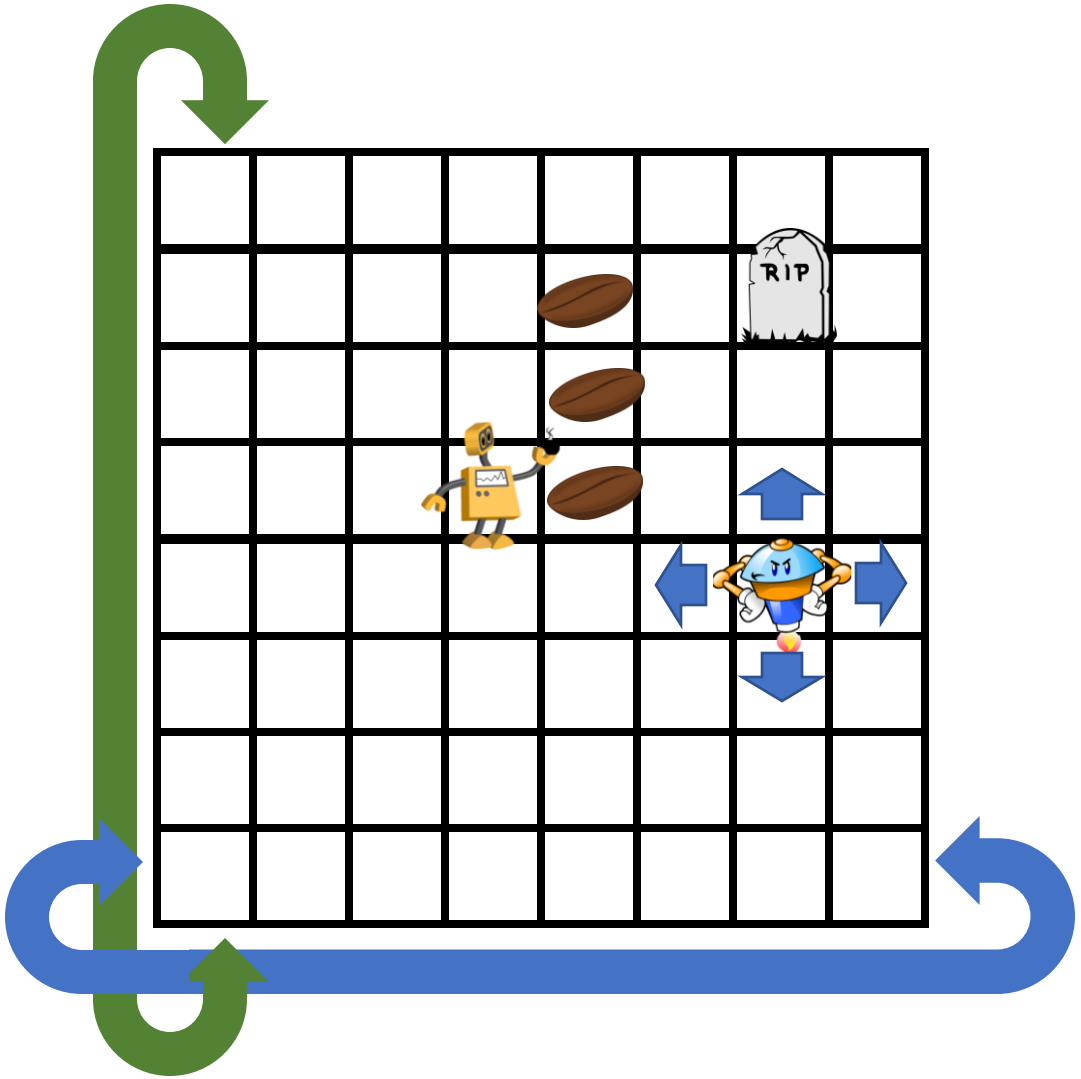


Understand the Problem

- **Observation**: It's hard to figure out what the program is supposed to do from its code
- Before looking at the code use existing documentation to understand the problem
- **Goal**:
  - Describe what the software is supposed to do using one whiteboard or large sheet of paper.
  - Why?

    Build a mental image of what is happening!
- Understanding the problem helps understand the reasons for the software's structure.

Example: TimSim (TimBots on DohNat)



- Simulation takes place on a grid (*DohNat*)
  - Edges wrap around
- Each grid element (*District*) can hold a robot (TimBot)
  - SpressoBot, ChickenBot, AngryBot, BullyBot
- The Bots can move around to different districts
- Bots need spresso to survive
- A round involves sensing, shooting, moving, battling, and harvesting
- TimSim runs the simulation for a number of rounds to see which bot survives.

Choose an Approach (Likely need to use both)

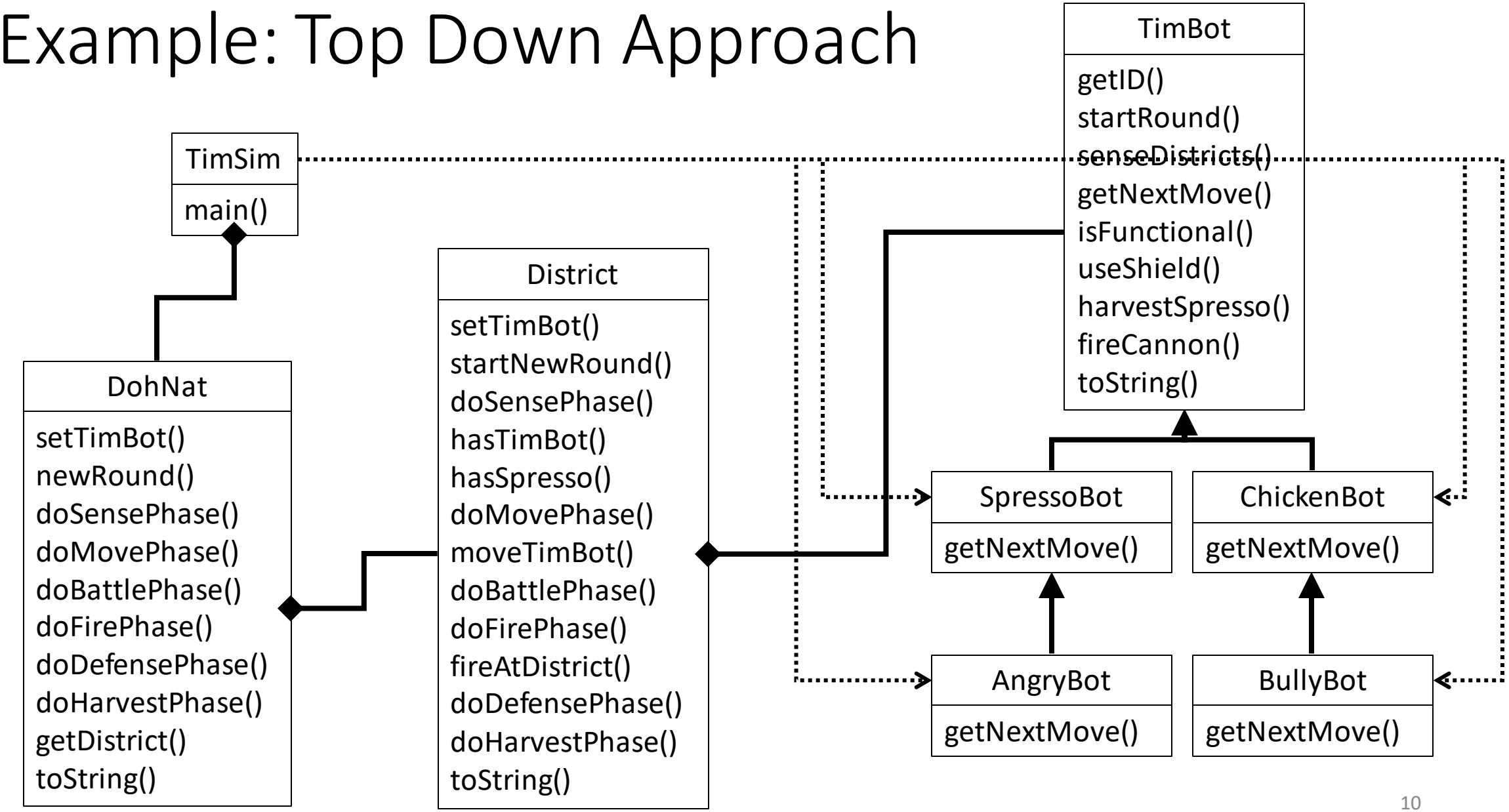
Top-Down Approach

- Create a UML diagram of the classes
- Identify the collaboration between them
- Identify the primary classes and the supporting classes
- Identify the public methods in the primary classes and understand those first along with the main method.
- Add annotations to your UML diagram, as you document the class interactions

Control-Flow

- Select some simple input examples
- For each input example do the following
- Start in the main() method
- Identify all classes used in the main method and the public methods being called
- Look at those public methods and identify the classes and methods they are calling
- Repeat, building an understanding of what the program is doing
- Keep notes as you go along

Example: Top Down Approach

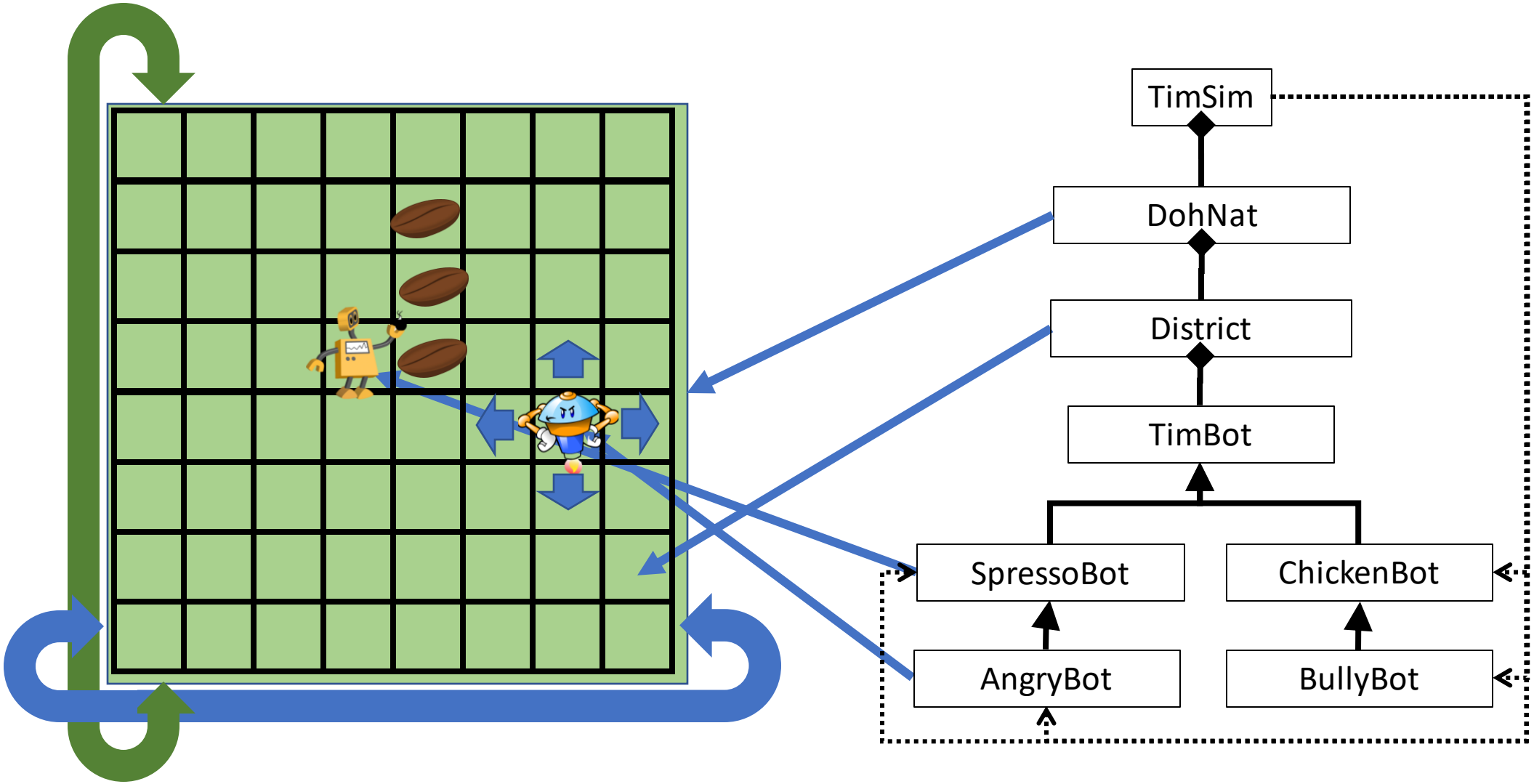


Relate the Problem Domain to the Classes

- If the software is well designed (and uses Object-Oriented Design) the classes should map to the problem domain.
- Use a combination of the problem description and the class descriptions to map classes to the problem domain
- Why do this?

  To understand the purpose of each class.

Example: Relating Problem Domain to Classes

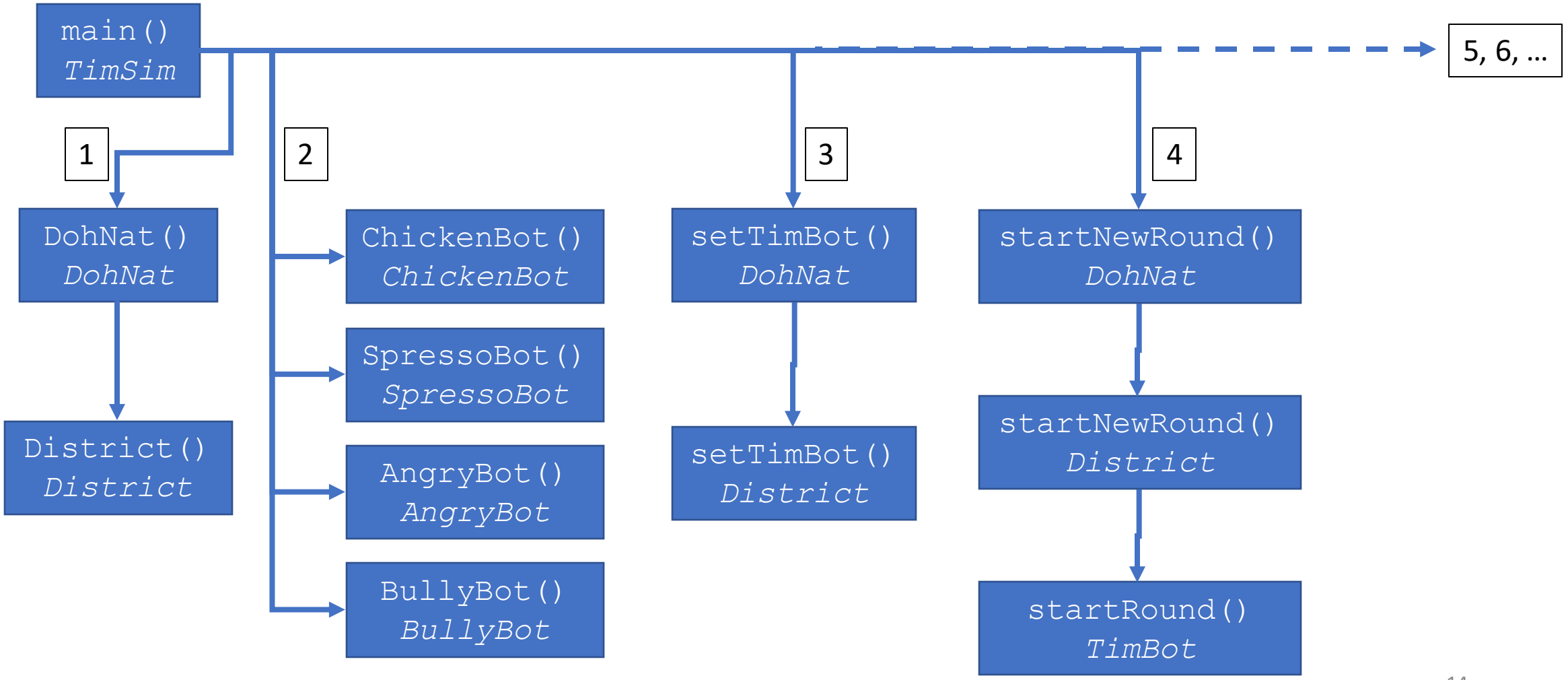


Control Flow

- Consider `main()` in `TimSim.java`
- Look for instantiation and calls to other classes

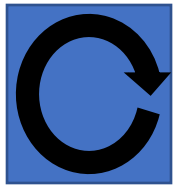
```
public static void main( String [] args ) {
  ...
  // Instantiate planet and array of timbots
  DohNat planet = new DohNat( rows, columns, jolts, growth );
  TimBot [] bots = new TimBot[numBots];

  // Load timbot configurations
  for( int i = 0; i < numBots; i++ ) {
    ...

    // Insantiate the corresponding bot object.
    switch( personality ) {
    case "chicken":
      bots[i] = new ChickenBot( id, energy );
      break;
    case "spresso":
      bots[i] = new SpressoBot( id, energy );
      break;
    case "angry":
      bots[i] = new AngryBot( id, energy );
      break;
    case "bully":
      bots[i] = new BullyBot( id, energy );
      break;
    }

  // Error checking (unnecessary)
  if( !planet.setTimBot( bots[i], x, y ) ) {
    ...
  }
```

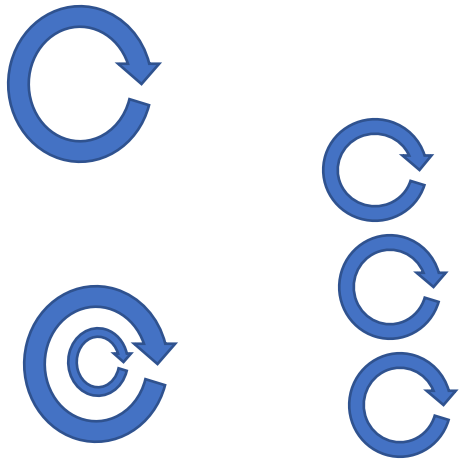
Example: Control Flow (TimSim)



Identify the Main Loop

- Most programs will have a main loop
  - Piece of code that loops and invokes the rest of the code
  - This is the code that dictates the high-level behavior of the program
- Variants:
  - Single big loop
  - Multiple big loops, corresponding to phases in a program
  - Nested loop
  - Event loop (implemented by a framework)
    - E.g., Java Swing

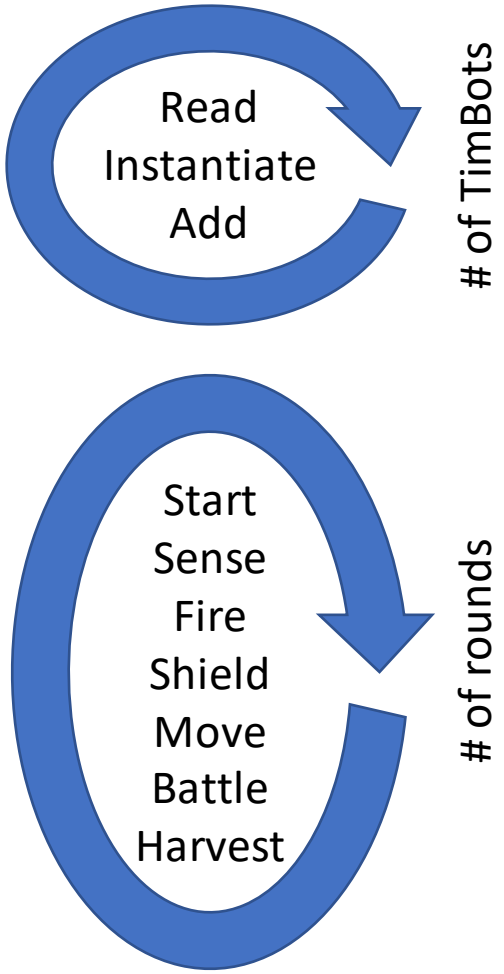




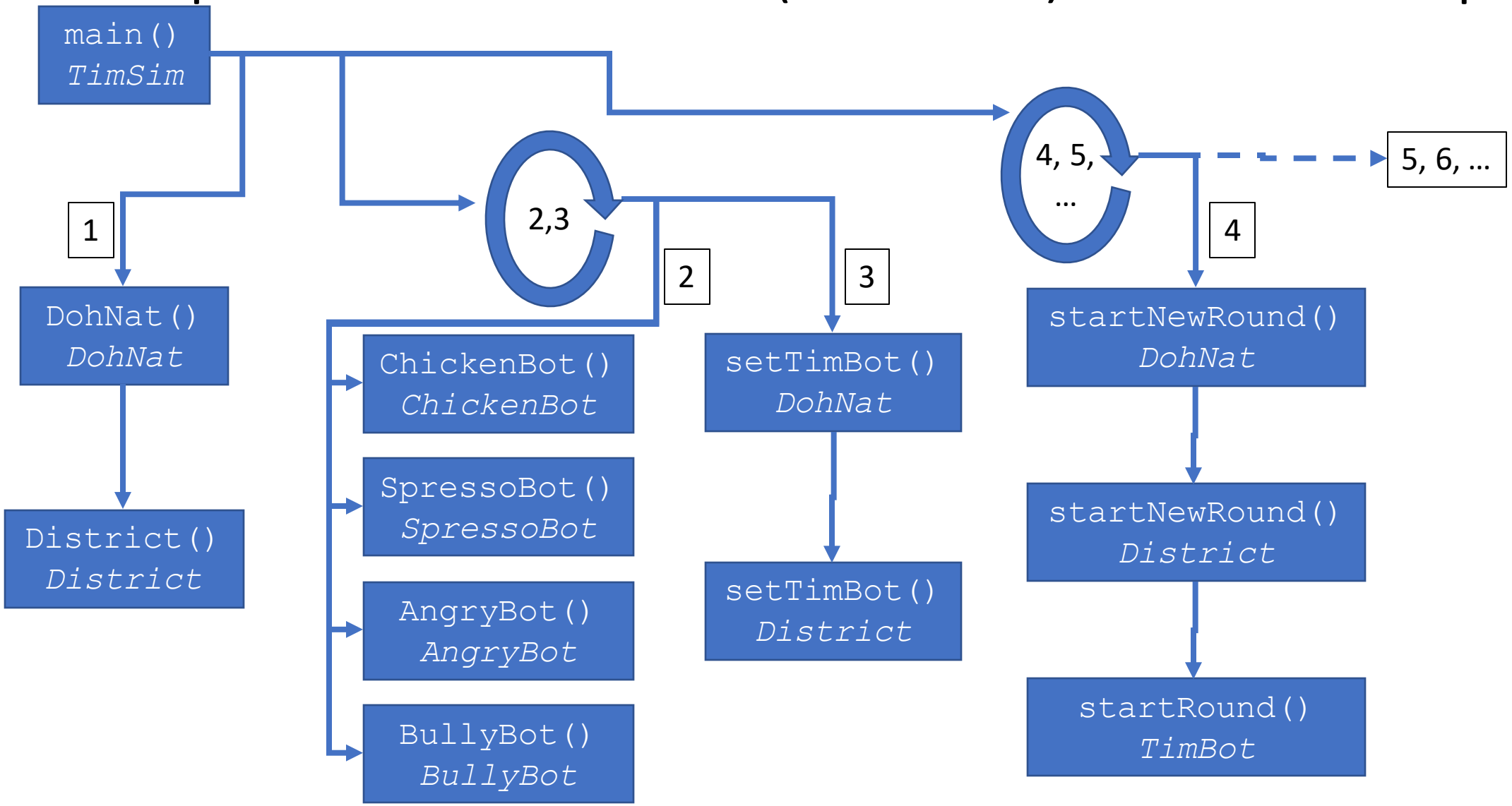
Example: Main Loops in TimSim

Two loops:

- Input loop
  - Read in a TimBot specification
  - Instantiate TimBot
  - Add to DohNat
- Round loop
  - Start round
  - Sense districts
  - Fire cannon
  - Use shield
  - Move
  - Battle
  - Harvest



Example: Control Flow (TimSim) + Main Loops



Other Ways to Comprehend Code

- Formal code reviews
  - Reading and going through the code together can quickly resolve misconceptions
- Code walkthroughs and other collaborative review processes
- Automated code extraction tools
- Debugging the code
  - One of the first traditional tasks for a new employee is to debug a piece of code
  - This forces them to learn the code they will be working on. ☺

Key 

Points

- Code comprehension is the process of understanding a codebase
- Typical approaches include
  - Analyzing the class structure
  - Analyzing the code flow

Image References

Retrieved January 29, 2020

- http://pengetouristboard.co.uk/vote-best-takeaway-se20/
- https://www.pngitem.com/pimgs/m/175-1757421_problem-clipart-transparent-clip-art-problem-solving-hd.png
- https://thumbs.dreamstime.com/b/needs-wants-folders-show-requirement-desire-showing-38165811.jpg
- https://www.researchgate.net/profile/Sujeet_Kumar4/publication/264231369/figure/fig3/AS:631868940627969@1527660703582/Class-diagram-of-SnW-module-In-C-implementation-four-classes-SnWT-SnWR-SnWHandler.png
- https://www.researchgate.net/publication/329431434/figure/fig3/AS:700644700721153@1544058122418/Embedded-software-control-flow-diagram-https-doiorg-101371-journalpone0208168g003.jpg
- https://www.researchgate.net/profile/Mohammad_Yarmand/publication/228932204/figure/fig6/AS:300672068145157@1448697215653/The-class-diagram-of-the-implementation-classes.png
- https://marketplace.eclipse.org/sites/default/files/20090806-cfgf-example-graphs.png