

Foodchain

An interactive ecosystem

Nancy Qian

Objective:

Explore and *stimulate* environmental scenarios

How do different variables affect a Foodchain?

What is the cause of Foodchain imbalance?

Which organism has the highest survival rate?

What and is there an optimal Foodchain?

What is in the Foodchain environment (initially)?

<ul style="list-style-type: none">• Animals<ul style="list-style-type: none">- Wolf- Rabbit	<ul style="list-style-type: none">• Plants<ul style="list-style-type: none">- Dandelion
--	---

Foodchain hierarchy

(Predator -> Prey)

Wolf->Rabbit->Dandelion

What are the *variables* in the ecosystem?

Weather condition (Rainfall): factor multiplied to plant variables in order to affect growth, birth, etc.

- Intensity (Range: 1-10)

Birth periods: periods of time between reproduction counted by **ticks***

- Wolf born period (Range: 1-400)
- Rabbit born period (Range: 1-400)

Birth rate: the percentage rate of successful reproduction every birth period

- Wolf born period (Range: 1-100%)
- Rabbit born period (Range: 1-100%)
- Dandelion born period (Range: 1-100%)

Exploration:

Scenarios with differing Independent
Variables

Scenario #1: Light Rainfall (weather condition)

Variables

- Weather condition= Lowest (1)
 - Born periods= Initial (200)

Description:

A situation in which a habitat experiences abnormally low levels of rainfall.

Results (Data):

- Video Data

Timestamps:

1. At 00:01:10, Dandelions die out
2. At 00:01:58, Rabbits die out
3. Lastly, at 00:02:10, Wolves die out



Conclusion:

- The dandelion died out first due to the lack of rain (born period impacted)
- This then caused the rabbits to die out from lack of food source, dandelion
- Lastly, the wolves die due to lack of rabbits
- A shortage of dandelion causes **imbalance** of the Foodchain and is not optimal

Real World Application

- An example of this could be the deserts or droughts
- Lack of rainfall or bodies of water
- **This is a problem because they:**
 - Plant growth depends on rainfall
 - Land becomes infertile
 - Does not support life (lack of water)

Scenario #2: Shortened Wolf born period

Variables:

- Weather condition= Initial (1)
- Wolf born period= Lowest (1)
- Rabbit born period= Initial (200)

Description:

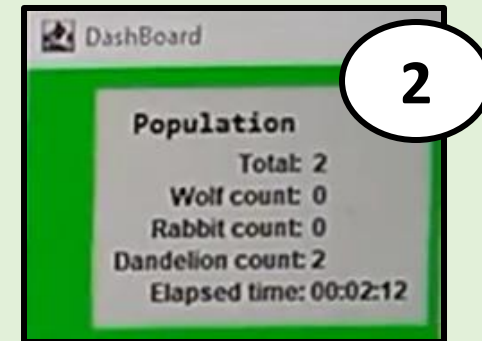
A situation in which the habitat's wolves reproduce at an abnormally high rate; period of time between each birth shortened

Results (Data):

- Video Data

Timestamps:

1. At 00:01:06, Rabbits die out
2. At 00:01:36, Wolves die out
3. Dandelions keep growing (may eventually die out)

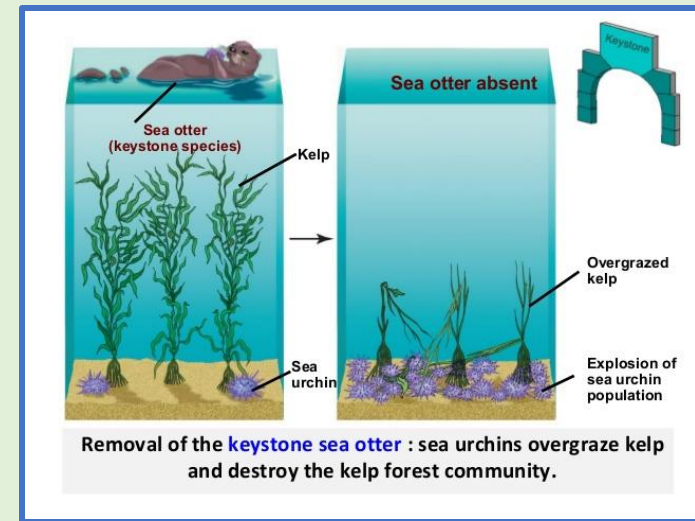


Conclusion:

- The rabbits died out first due to the amount of wolves (produced by the low born period)
- This then caused the wolves to die out from lack of food source, rabbits
- The dandelion aren't fatally affected by this since their predator rabbit died first
- A large production of wolves causes **imbalance** of the Foodchain and is not optimal

Real World Application

- An example of this could be keystone predators
- **This means:**
 - Predators introduced to moderate prey population and competition



- **This is could be a problem because they:**
 - Could hunt prey population to extinction

Scenario #3: Shortened Rabbit born period

Variables:

- Weather condition= Initial (1)
- Wolf born period= Initial (200)
- Rabbit born period= Lowest (1)

Description:

A situation in which the habitat's rabbits reproduce at an abnormally high rate; period of time between each birth shortened

Results (Data):

- Video Data

Timestamps:

1. At 00:01:06, Dandelions die out
2. At 00:01:36, Rabbits die out
3. Lastly, at 00:02:06, Wolves die out



Conclusion:

- The dandelion died out first due to the amount of rabbits (produced by the high born rate)
- This then caused the rabbits to die out from lack of food source, dandelion
- Lastly, the wolves die due to lack of rabbits
- A large production of rabbits causes **imbalance** of the Foodchain and is not optimal

Real World Application

- An example of this could be the [European rabbits in Australia](#)
- **European rabbits:**
 - Have a lack of natural predators
 - Can give birth to more than **four litters** a year with as many as **five kits** (baby rabbits) each (**high birth rate**).
- **This is a problem because they:**
 - Drive out native species from their homes
 - Compete for food and other resources
 - Loss of plant biodiversity

Exploration Takeaways

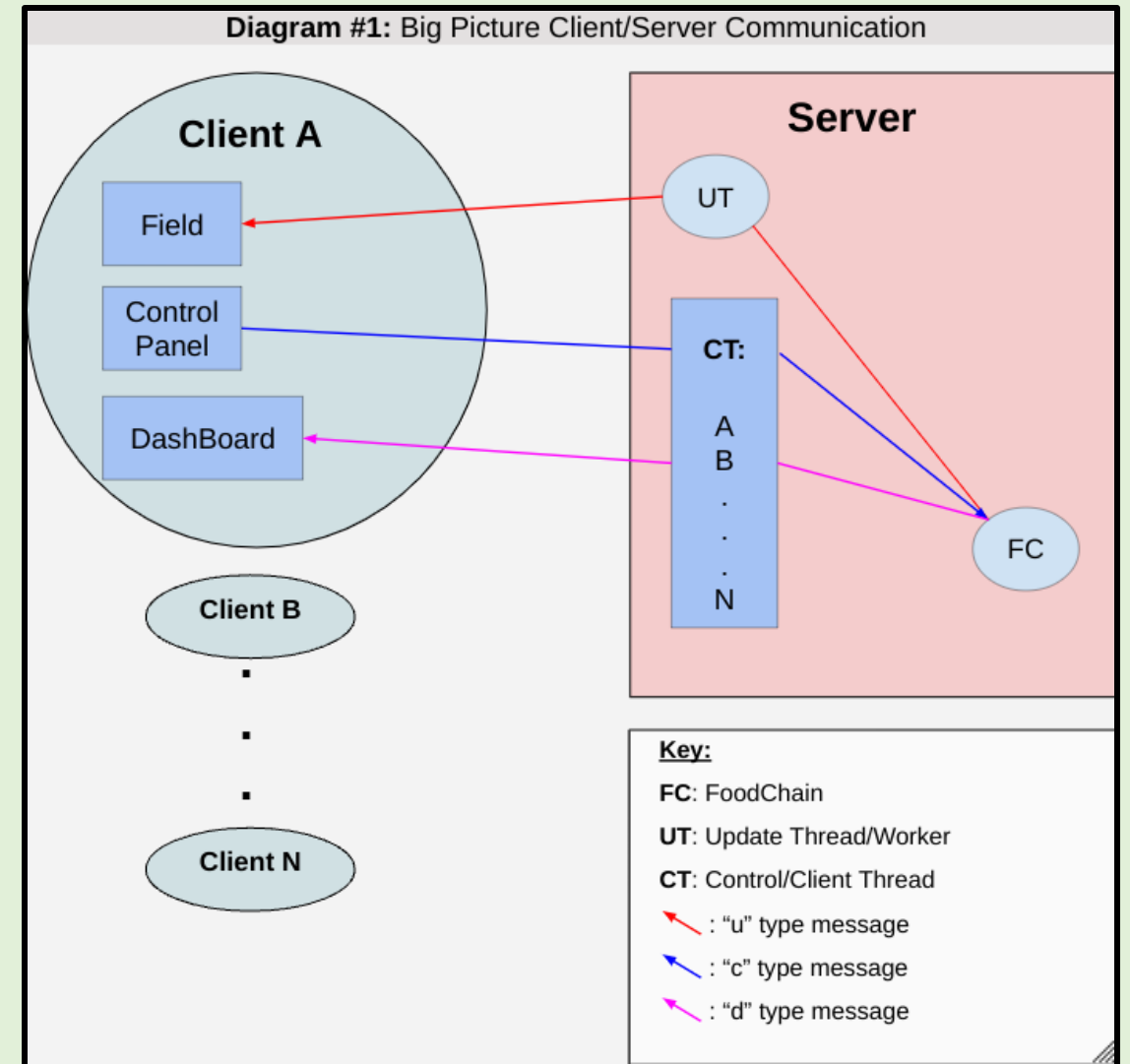
- Foodchain imbalance could be caused by:
 - Due to drastic changes in the variables, some part of the Foodchain is eliminated, the entire chain of predators before it that relied on it for food would be affected
 - **Example from our Foodchain:**
 1. Wolf->Rabbit (high birth rate)->~~Dandelion~~ (all eaten by Rabbit; extinct)
 2. Wolf->~~Rabbit~~ (high birth rate but now; no food source; all eaten by Wolf; extinct)
 3. ~~Wolf~~ (no food source, extinct)
- Introduction of **non-native species** with extremely high birth rates, such as the rabbit, could end up causing plant species, like the dandelion, **to go extinct**, which causes the rabbits to go extinct as well, if dandelions were a crucial part of their diet; this also impacts the wolves who hunt the rabbits for food (the WHOLE Foodchain affected)

Stimulation:

Creating the Foodchain

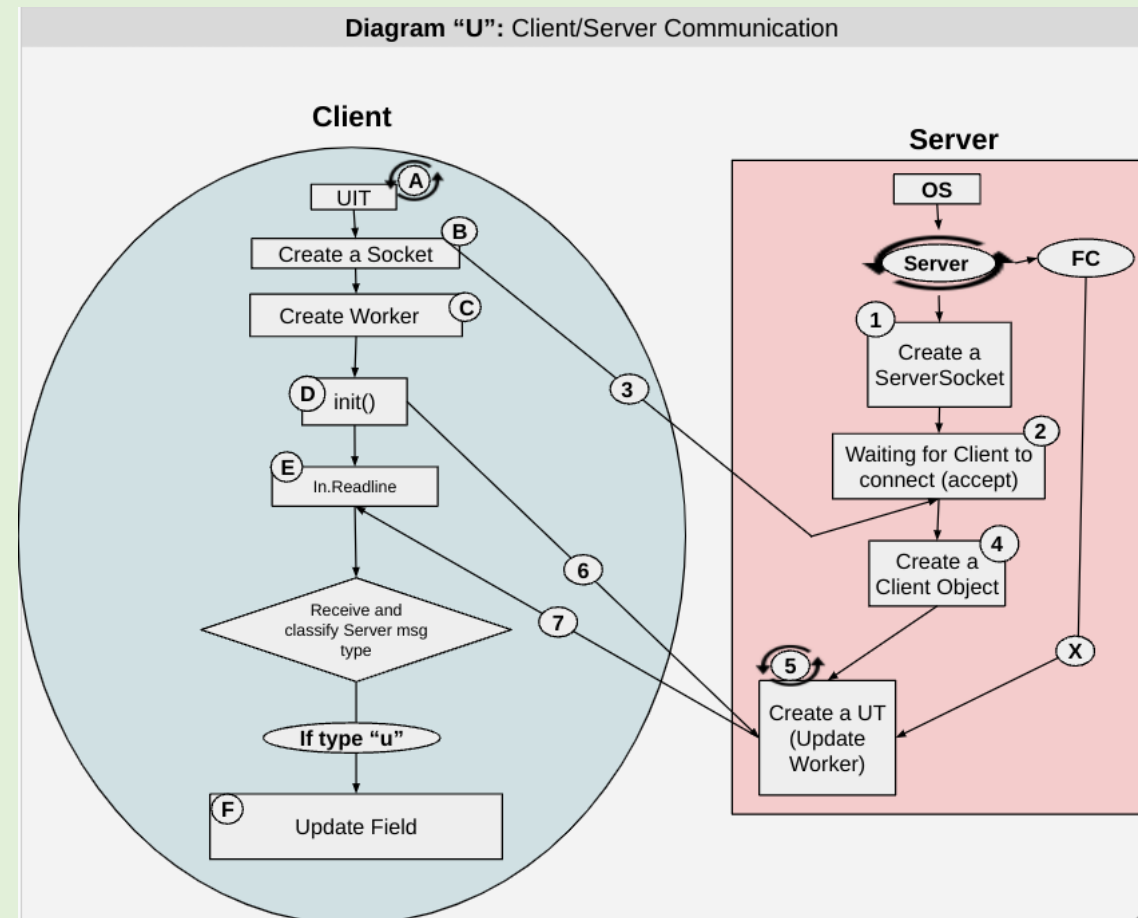
The Broader Picture

- Create a **Server** with a running environment
- Multiple **Clients** from different platforms that can connect to Server to view environment



"U" messages: Update Field UI

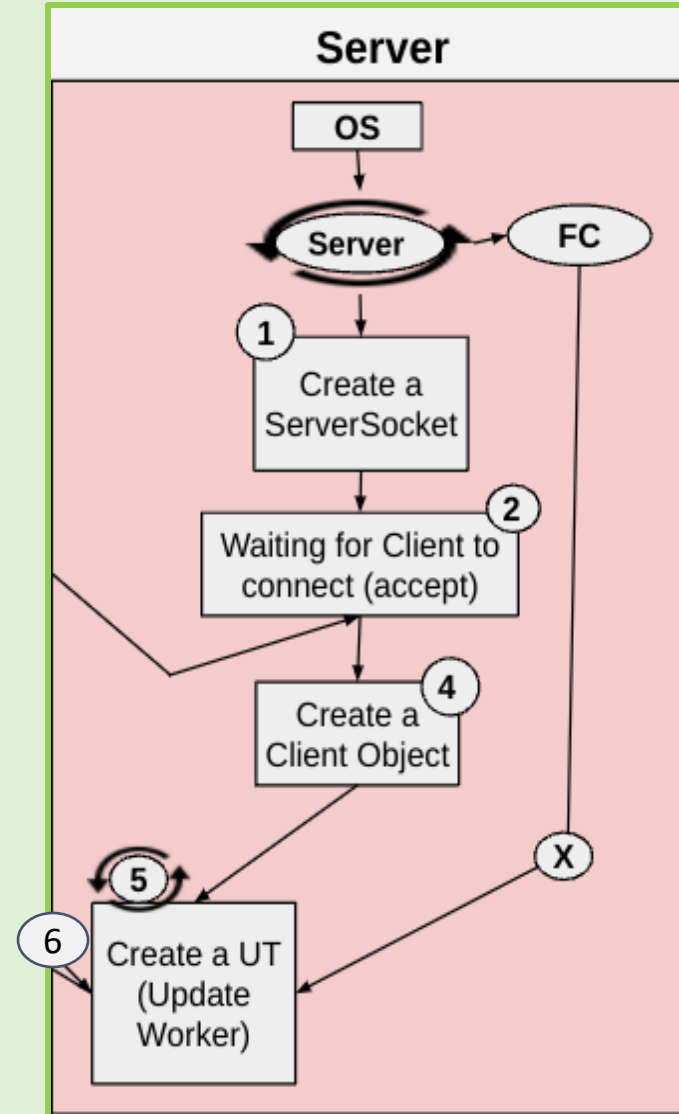
Server's Update Worker constantly pushes values to update FoodChainField (state of animals, position, environment)



"U" messages (Update Field UI): Server side

Tool Tips:

- 2. Call accept of ServerSocket (block Server control)
- 7. Constantly broadcast to all CT in Client List
- X. Update Worker constantly updates using FoodChain class's Life List (thread list)



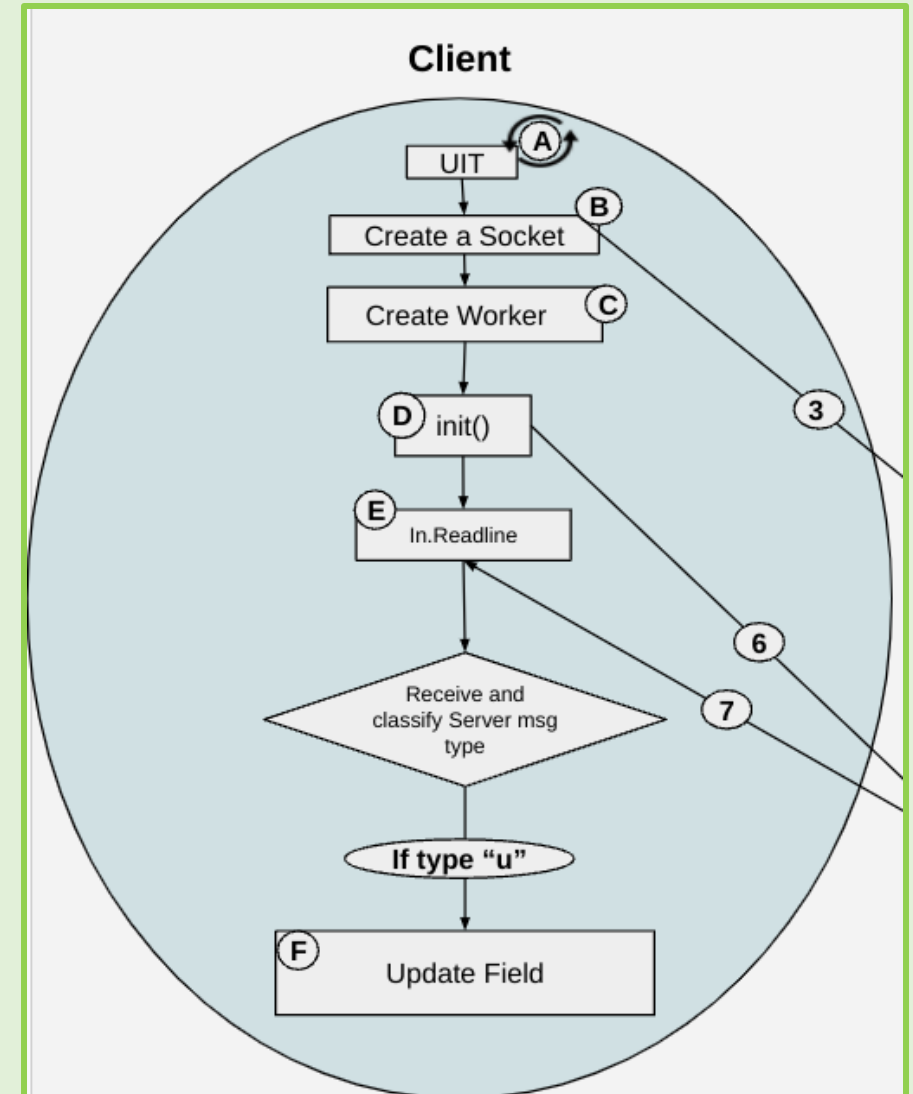
"U" messages (Update Field UI): Client side

Tool Tips:

A: create a UI Thread

B: create a Socket (unblocks Server that was waiting for Client response)

C: create a Worker Thread for Client



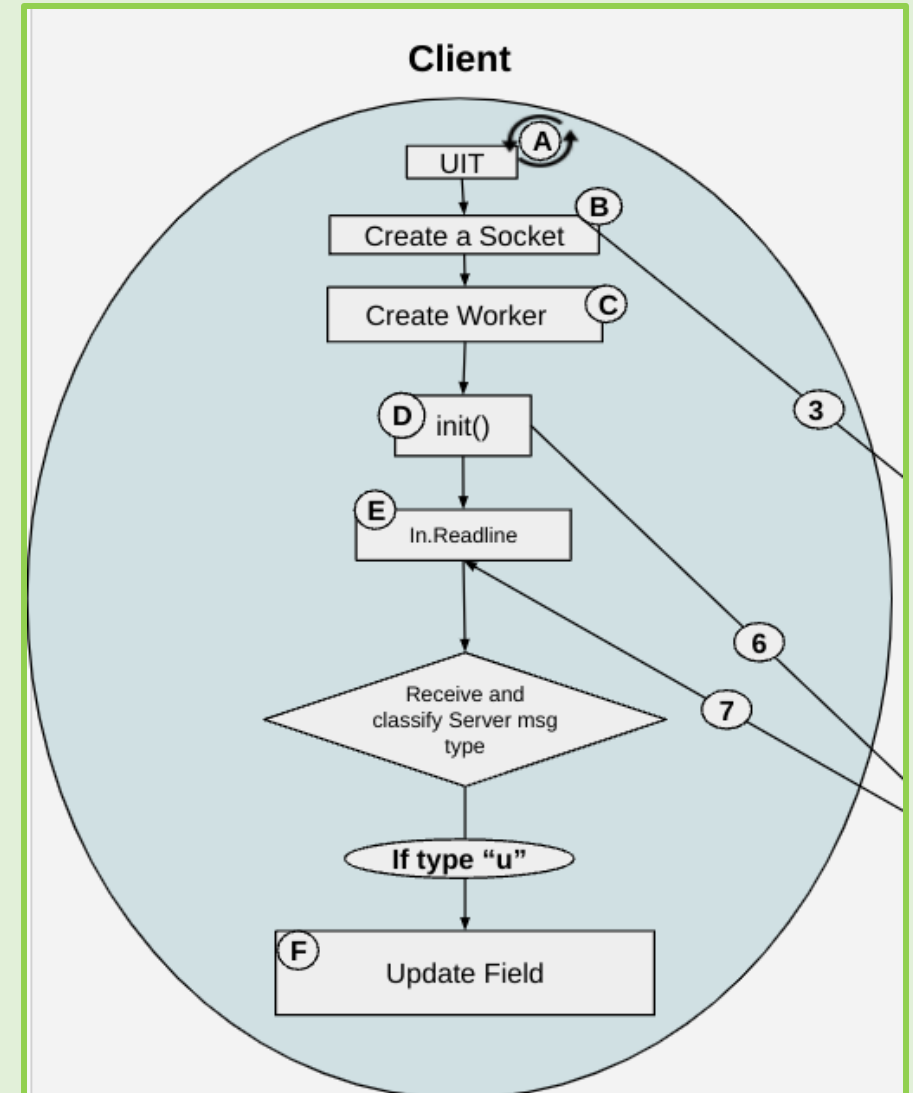
"U" messages (Update Field UI): Client side

Tool Tips:

D(6): call init() method to send (out) key from Client that need initial slider values

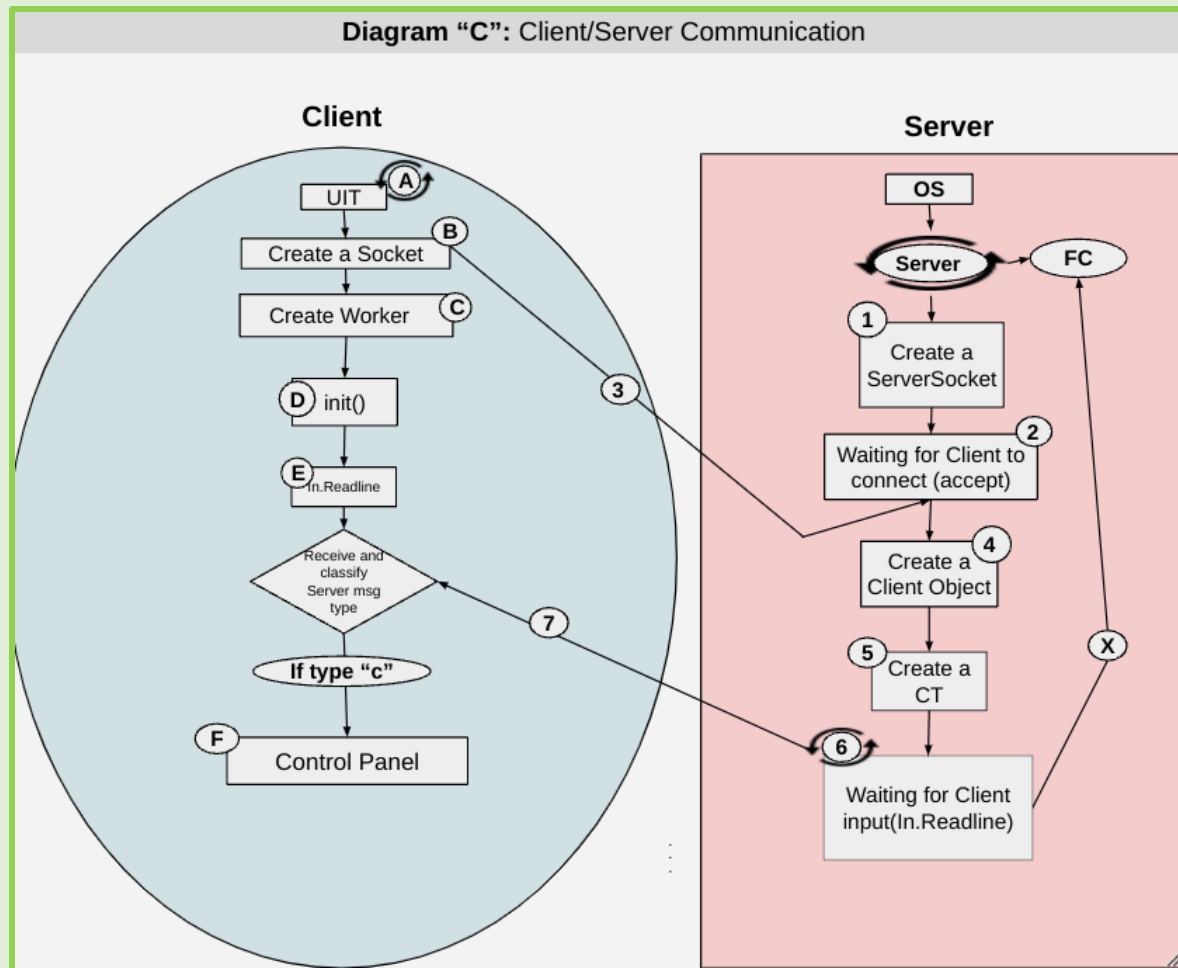
E: waiting for msg from Server (in)

F: Update FoodChainField UI



"C" messages: Update Control Panel

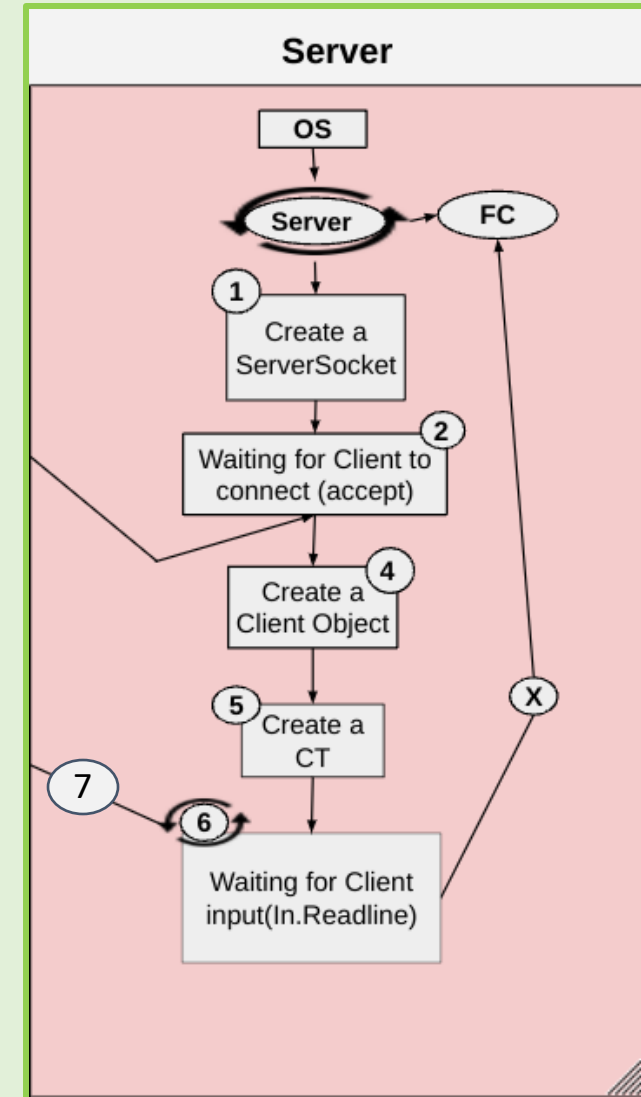
Client's Control Panel sends update request to Server's Client Worker
(updates change of values on sliders)



"C" messages (Update Field UI): Server side

Tool Tips:

- 2. Call accept of ServerSocket (block Server control)
- 5. Create a Client Thread
- 6. Client Thread waits for Client to send Control request
- 7. Client Thread sends values from FoodChain to Client
- X. Retrieves values requested by Client from FoodChain



"C" messages (Update Field UI): Client side

Tool Tips:

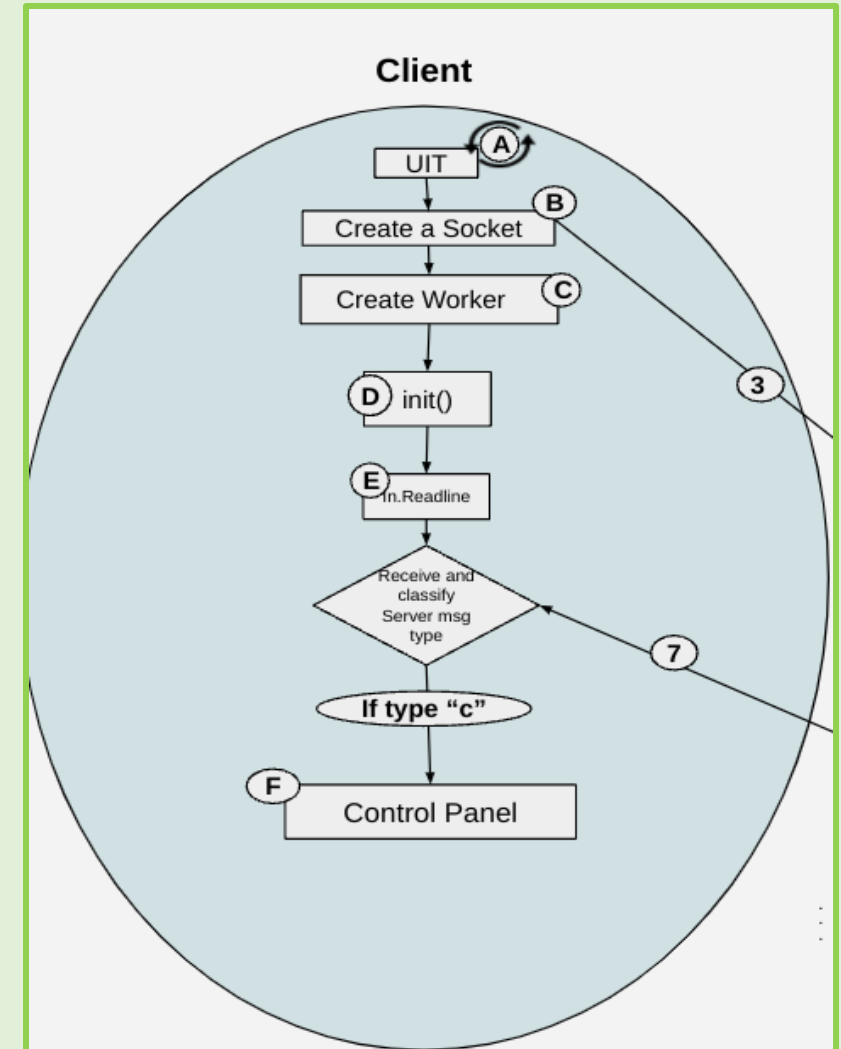
A: create a UI Thread

B: create a Socket (unblocks Server that was waiting for Client response)

C: create a Worker Thread for Client

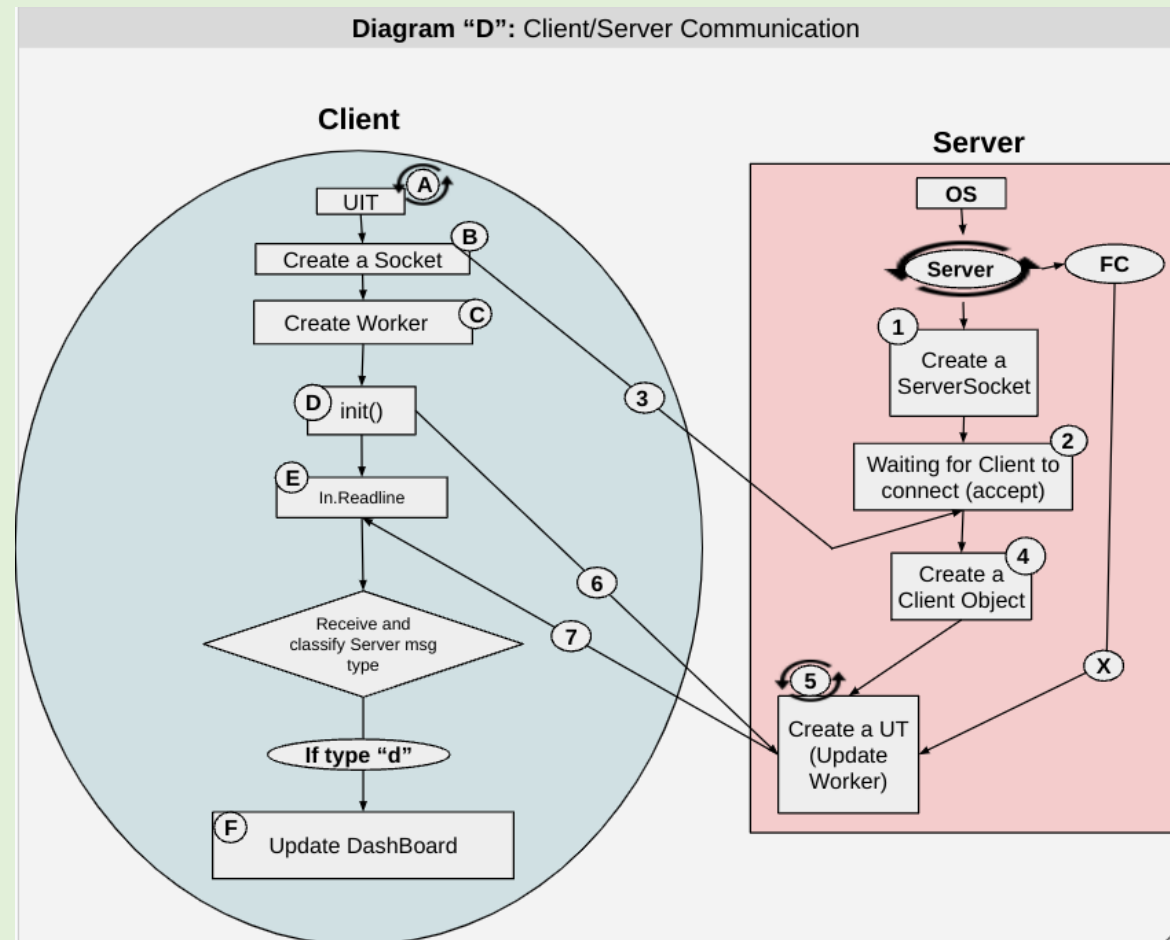
E: waiting for msg from Server (in)

F: Update Control Panel (values in sliders)



"D" messages: Update Control Panel

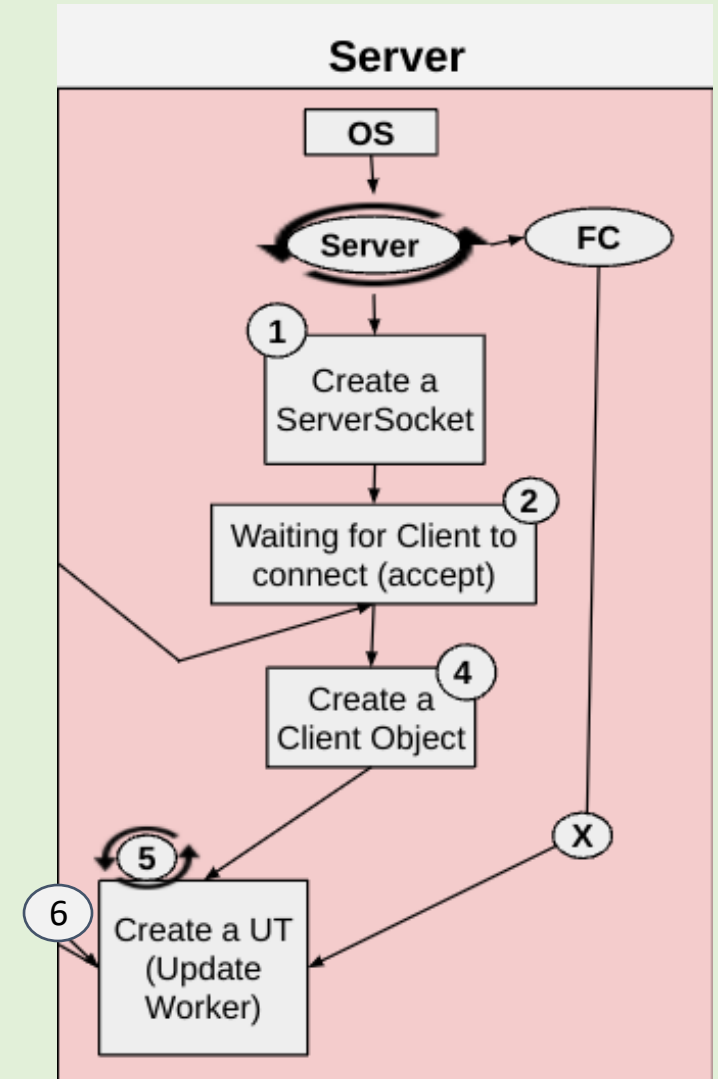
Client's DashBoard sends update request to Server's Client Worker (constantly pushing values displayed on DashBoard, population, weather, etc.)



"D" messages (Update Field UI): Server side

Tool Tips:

2. Call accept of ServerSocket (block Server control)
4. Create a Client Thread
5. Client Thread waits for Client to send Dashboard request
6. Client Thread sends Dashboard values from FoodChain to Client
- X. Retrieves values requested by Client from FoodChain



"D" messages (Update Field UI): Client side

Tool Tips:

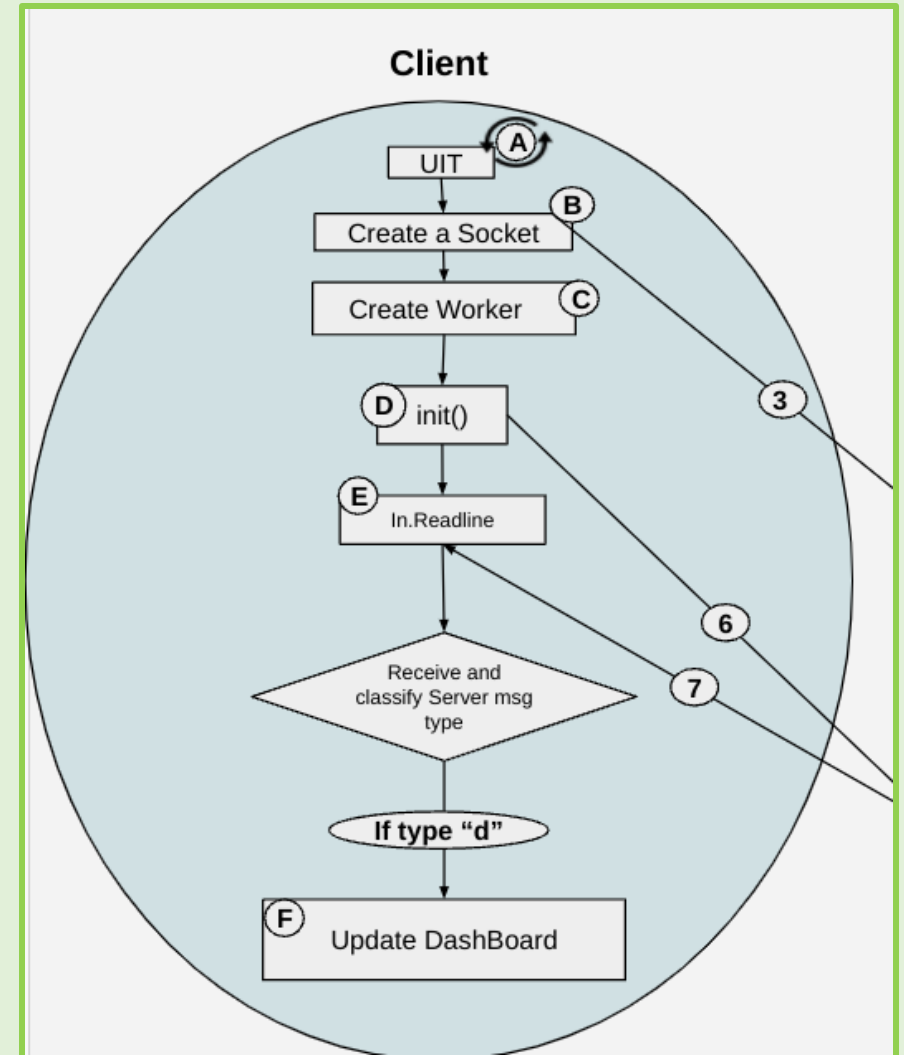
A: create a UI Thread

B: create a Socket (unblocks Server that was waiting for Client response)

C: create a Worker Thread for Client

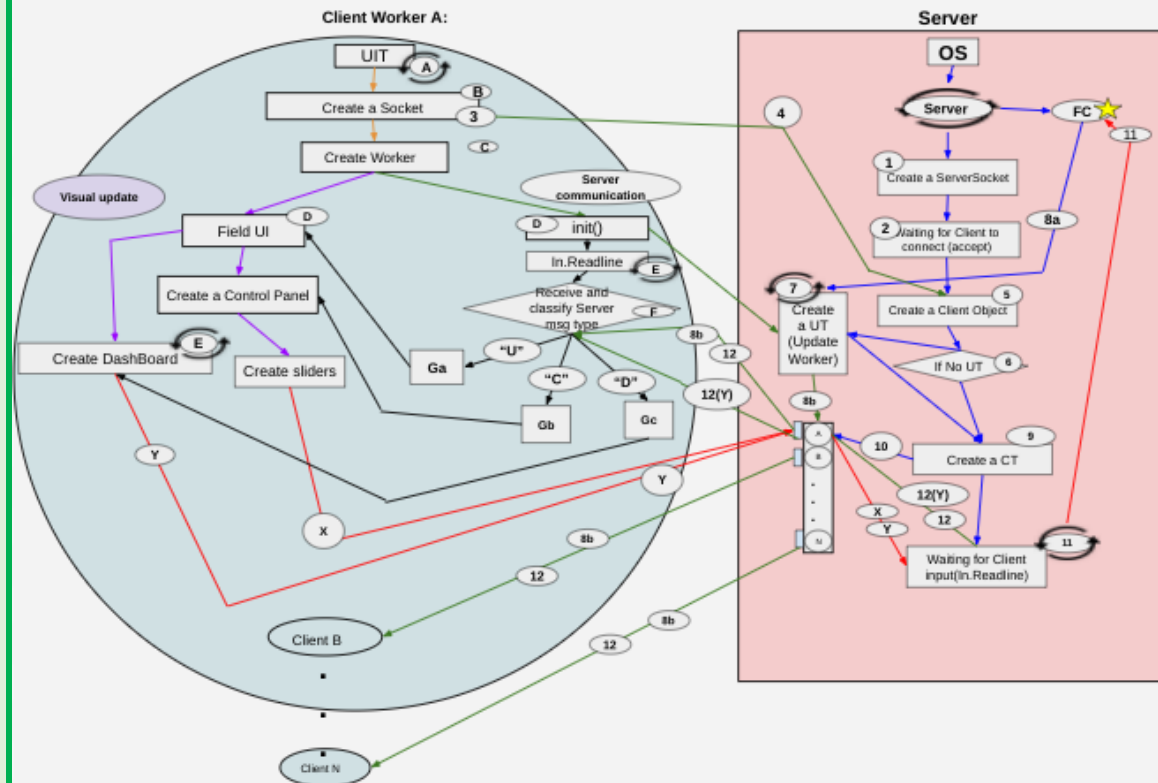
E: waiting for msg from Server (in)

F: Update Dashboard (values in labels)



The Whole Picture: Client/Server Communication

Diagram #1: Client/Server Communication



Tool Tips:

2. Call accept of ServerSocket
4. Send a connection request to unblock waiting server caused by accept()
6. Add the ClientThread or CT (socket/address) to list
7. Client's Worker receives changed fields and updates
 - *Update thread continuously broadcast LifeList to Client's Worker
 - 8a. Update Worker updates using FoodChain class's Life List (thread list) using getLifeList
 - 8b. Constantly broadcast to all CT in Client List
11. CT waits for input from Client
11. Server uses apply() to change fields to match Client fields
12. Call broadcast() method to all Client Workers
 - *Y. CT only out.println to its own client/socket

A: create a UI Thread
C: create a Worker Thread for Client

Visual update







E: create DashBoard as Runnable, then a thread that sends update requests to Server when opened

Server communication

D: call init() method to send (out) key from Client that need initial slider values
E: waiting for msg from Server (in)
Ga, Gb, Gc: call update()/setFields() to set Server values to Client values for Field UI/Control Panel sliders/Dashboard

Ha: update field values to Field
Hb: repaint Control Panel
X: User action adjusting slider value
detected by slider's ChangeListener, sent as
msg with toJson() to Server's CT through
socket

Key:

-  : socket
-  : thread (runnable)
-  : see Diagram #2
-  : communication between Client/Server
-  : User action procedure
-  : Client procedure
-  : Client (Visual update) procedure
-  : Client (Visual update) procedure

Java Application

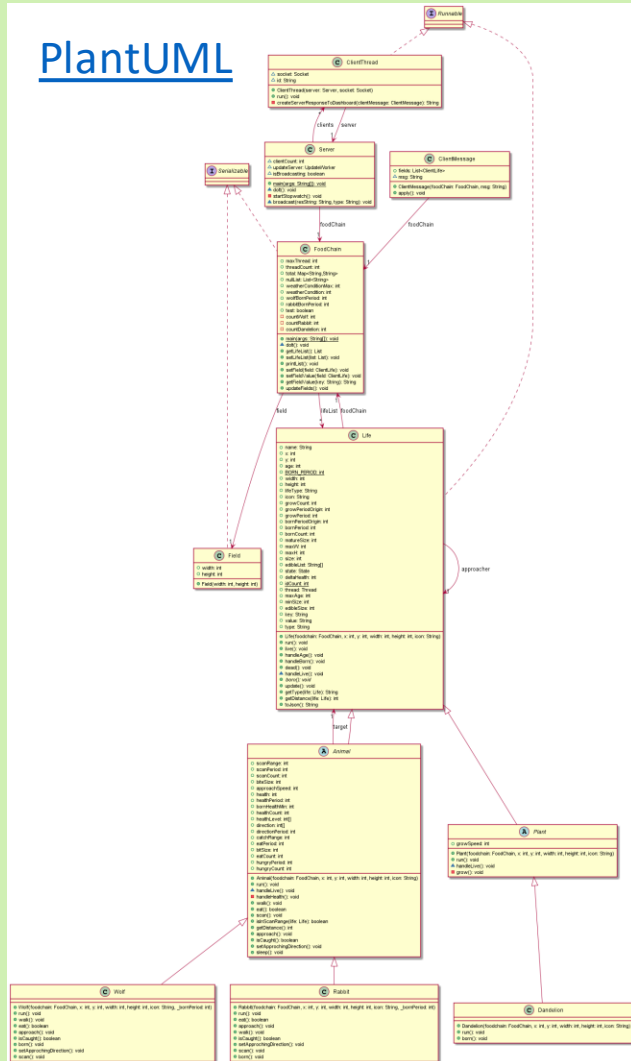
OOP Programming

- Inheritance
- Encapsulation
- Polymorphism

Inheritance

The system of superclasses and subclasses in which subclasses can inherit superclasses' accessible fields and methods

PlantUML



WHY?

- less code needed; parent class code can be reused in child class
- reduce redundancy
- *cleaner code
- Overriding: methods with identical "signatures" in superclass and its subclass, but polymorphism narrows it down to lowest subclass

Inheritance examples

Dandelion class has
superclass of Plant
class

```
6
7 public class Dandelion extends Plant{
8
9     public Dandelion(FoodChain foodchain, int x, int y, int width, int height
10         super(foodchain, x, y, width, height, icon);
11
12         growSpeed = 2;
13         size = 10;
14         this.bornPeriod = 250;
15         Random rand = new Random();
16         this.bornCount = 20+rand.nextInt(Math.max(bornPeriod, 2));
```

With inheritance,
Dandelion class can
inherit and reuse Plant
class' constructor

Encapsulation

the process of wrapping or hiding data from other classes

WHY?

- maintenance of the code becomes easier and cleaner
 - *easy to change
- Setting fields and methods private is also important in efficiency.
 - *prevents other classes to access

Encapsulation example

An example would be to use getters, setters, and adders:

```
177 @Override
178 public void born(){
179     if (health < bornHealthMin) {
180         return;
181     }
182     Life life = new Rabbit(this.foodChain, this.x+1, this.y+1, maxW/2, maxH/2, icon, bornPeriod);
183     if (foodChain.nullList.size() > 0) {
184         try {
185             foodChain.setLife(Integer.parseInt(foodChain.nullList.get(0)), life);
186         } catch (java.lang.IndexOutOfBoundsException e) {
187             foodChain.addLife(life);
188         }
189     } else {;
190         foodChain.addLife(life);
191     }
192     life.thread = new Thread(life);
193     life.thread.start();
194     foodChain.threadCount++;
195     state = State.NORMAL;
196 }
197 }
```

```
64
65 public void setLife(int i, Life life) {
66     lifeList.set(i, life);
67 }
68
69 public void addLife(Life life) {
70     lifeList.add(life);
71 }
72
73
74 public void getLife(int i) {
75     lifeList.get(i);
76 }
77
78 }
79
80
```

Call set(), add(), get() in respective voids instead of multiple times

Polymorphism

: an action with multiple forms

WHY?

- One variable name can be used to store variables of many data types

- Overloading: methods with different "signatures" (name, parameters, return type) in terms of parameters (different number, type, or both)

Ex. : `void A(int i)` vs `void A()` vs `void A(int i, int ii)`

- Overriding

Polymorphism examples

Life has method live() which calls handleLive(). Class Animal and Plant both extend Life: polymorphism allows them to **override** handleLive() in their superclass. This is an example of one action with many forms.

```
31 public void live() {  
32     while (state!=State.DEAD) {  
33         try{Thread.sleep(100);}catch (InterruptedException e) {}  
34         handleAge();  
35         handleLive();  
36         handleBorn();  
37     }  
38     foodChain.threadCount--;  
39 }  
40 }
```

```
37  
38 void handleLive() {  
39  
40     switch (state) {  
41         case NORMAL:  
42             walk();  
43  
44             break;  
45         case SLEEP:  
46             sleep();  
47             break;  
48         case SCANNING:  
49             scan();  
50             break;  
51         case APPROACHING:  
52             approach();  
53             break;  
54         case EATING:  
55             eat();  
56             break;  
57         case DEAD:  
58             dead();  
59     }
```

```
16  
17 void handleLive() {  
18     switch (state) {  
19         case GROW:  
20             grow();  
21             break;  
22         case DEAD:  
23             dead();  
24             break;  
25         case BORN:  
26             born();  
27             break;  
28         default:  
29             break;  
30     }  
31 }  
32 }
```

Polymorphism examples

```
for (int i=0; i<initNumWolf; i++) {
    Life life = new Wolf(this, (50+rand.nextInt(maxW)), (50+rand.nextInt(maxH)), 20, 20, "wolf.png", -1);
    lifeList.add(life);
    life.thread = new Thread(life);
    life.thread.start();
    threadCount++;
}

for (int i=0; i<initNumRabbit; i++) {
    Life life = new Rabbit(this, (50+rand.nextInt(maxW)), (50+rand.nextInt(maxH)), 15, 15, "rabbit.png", -1);
    lifeList.add(life);
    life.thread = new Thread(life);
    life.thread.start();
    threadCount++;
}

for (int i=0; i<initNumDandelion; i++) {
    Life life = new Dandelion(this, (50+rand.nextInt(maxW)), (50+rand.nextInt(maxH)), 27, 27, "dandelion.png");
    lifeList.add(life);
    life.thread = new Thread(life);
    life.thread.start();
    threadCount++;
}
```

Why not use polymorphism for everything?

Polymorphism would be suitable for mass production of Life Objects by calling born() method of Life by looping through the LifeList (list of Life objects). However, in this scenario, individually instantiating was more reasonable (as user would set their desired initial amount of each Life type in the arguments).

Appendix

- Ticks (Slide 4): an iteration through the class, Life's live() method