# Fish Tank Simulation

The basic design is of a **Tank** containing **Creatures** on four **Levels** - the surface, a top level, a middle level and a bottom level. (The surface is generally reserved for dead Creatures.)

## Communication between objects

Time passes by in intervals. When the user sends a '*pass_time*' command to the **Tank**, this ripples through from the Tank, through to the Levels and their Creatures, then back out again. This facilitates functions such as reporting on events (a snail died, a piranha ate another fish, a fish could not eat etc), and transferring dead fish to the surface Level.

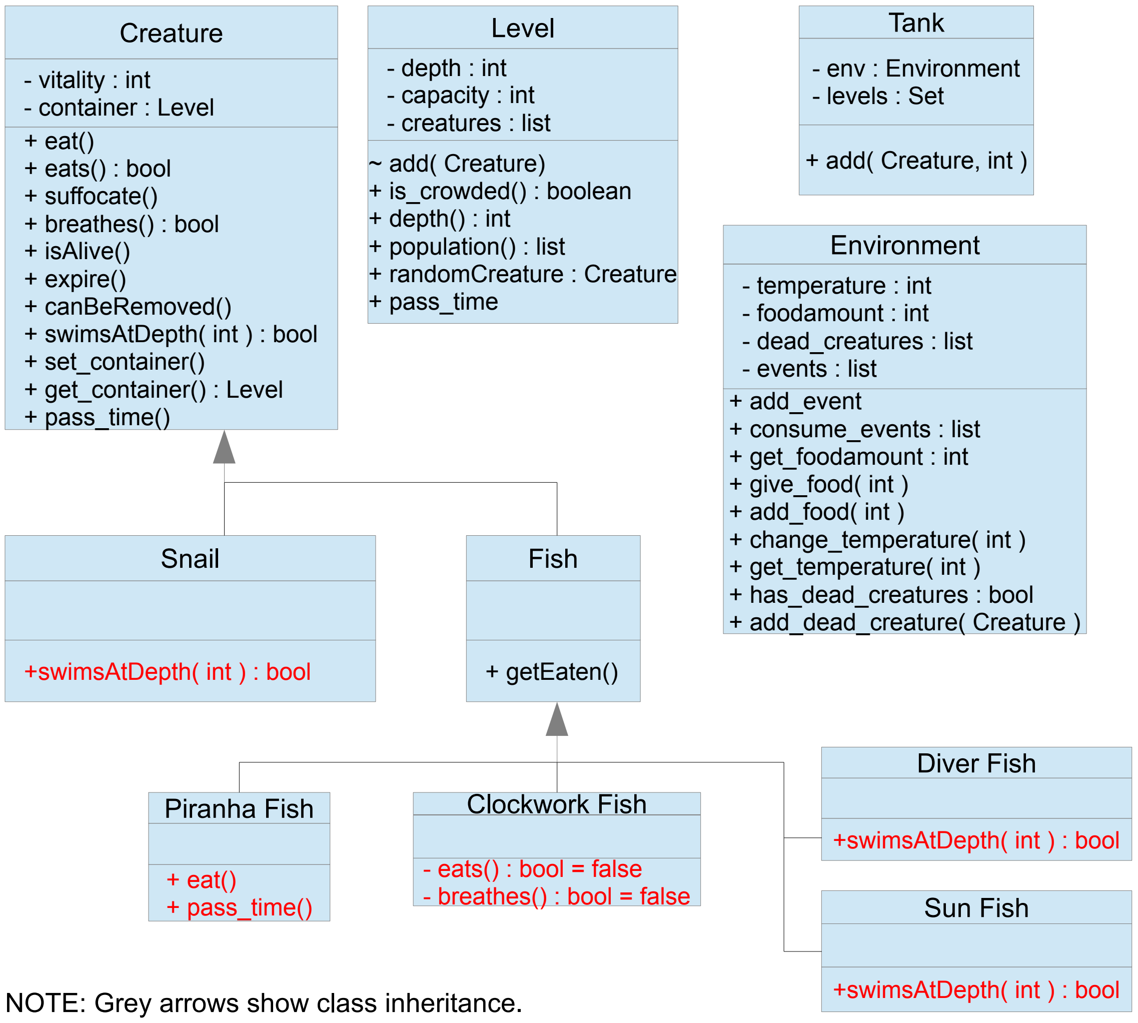
## Creatures

The Creature has methods for its interaction with its containing Level and the Environment. All creature objects have this as their base class, and its subclass, Fish, is the ancestor of all types of fish. This makes it easier to implement certain characteristics that are common to certain types of fish. Other fish that do not share a characteristic can simply override the behaviour e.g. for a ClockworkFish, its '*eats*' property is overridden to 0, so it will not *eat* any food when it is called upon to *pass_time*. The PiranhaFish will 'eat' fish food (if there is any) as well as eat another Fish in the same time interval (again, if there are any other Fish).

A Creature also possesses a *vitality* property which increases if it *eat*s and there is food in the Environment. Its *vitality* generally counts down with each *pass_time* call and it will *expire* (die) when it reaches zero. Generally, a Creature is transferred to the surface Level when it dies, unless it is eaten by a PiranhaFish.

When a Creature is introduced to a Level, its internal '*container*' property is set, so it can check if the Level *is_crowded* and whether to *suffocate*/*expire* (die), if it *breathes*.

### Class Diagram



NOTE: Grey arrows show class inheritance.

## User Actions
In the 'user_interface.pl' interactive script provided, the user is able to carry out the simulation as described, following the menu prompts.

The user can:
- Do nothing/let an interval pass
- Add a fish or snail (have to specify the Depth)
- Add fish food
- Increase/decrease the temperature
- Quit/halt the simulation

## Considerations leading to final decision
1. Fish were given a 'vitality' level so they could die rather than the user simply adding more and more creatures until the fish tank is full, which I thought was a bit unrealistic. (Also to give them another way to die other than being eaten.)

2. I could have used multiple inheritance and separated out the Creatures even further as Breathing and NonBreathing, but did not do so, and left these as simple properties, as the vast majority of the Creatures eat and breathe, and the non-conforming creatures simply overrode this behaviour. In future, development of interfaces/superclasses for Breathing creatures could be possible if more non-breathing Creature types are developed.

3. The Environment class was not in the original design, but I included it in order to pass information from Tank <-> Level <-> Fish, outside-in and vice versa. In particular, dead fish need to be subtracted from their current level and added into the 'surface' level. Also, PiranhaFish need to be made aware of the temperature. There are other ways of doing this (e.g. a publish-subscribe model), but this was a simple and fairly elegant way.

4. I added a 'container' property for a Creature so it could inspect the rest of the level. This was useful for PiranhaFish, who could select a random Fish to eat.

## Public methods
Almost all of the methods are fine to be called by other classes within the project (the ones that aren't are prefixed with underscores). However, for <u>users</u> of the project e.g. within a test script or the user interface, only the following methods should be used (as per user_interface.pl):

- the constructors for **Tank**, and the **Creature**s (Snail, ClockworkFish, DiverFish, PiranhaFish, SunFish)
- **Tank**: *add_food*, *change_temp*, *add*( Creature, level specifier ), *pass_time*

The following can be used for debugging purposes:
- **Tank**: *levelAt*( 'SURFACE' | 'TOP' | 'MIDDLE' | 'BOTTOM' )
- **Level**: *population*
- **Creature**: *isAlive*

(Note: if an extra debug parameter is passed to *pass_time* it will print out the contents of the tank (see example at end), as well as any events that happen during the interval.)

## Assumptions
1. A creature placed at a certain depth stays at that depth for the rest of its life.
2. Dead fish/snails do not decompose. (Simplicity)
3. Only the quantity of fish food is tracked, not the depth -- so if there is food in the tank, any creature that can eat, does, subtracting one unit for each creature that eats. (Simplicity)
4. A level is 'too crowded' if the number of creatures is greater than its given Capacity.

## Unresolved Issues (outside of assumptions)
1. Do dead fish poison live fish on the surface?

Example of a debug display of pass_time:

```
[ SURFACE ] 1 dead creature(s) now.
| Depth: 0
| Crowding: 1/10
| Population: 1 dead PiranhaFish
  [SunFish] no food...
[ TOP ] 0 dead creature(s) now.
| Depth: 10
| Crowding: 1/10
| Population: 1 SunFish
[ MIDDLE ] 0 dead creature(s) now.
| Depth: 30
| Crowding: 0/10
| Population:
  [DiverFish] no food...
  [DiverFish] no food...
  [DiverFish] no food...
  [DiverFish] no food...
  [DiverFish] no food...
  [DiverFish] no food...
  [Snail] no food...
[ BOTTOM ] 0 dead creature(s) now.
| Depth: 50
| Crowding: 7/10
| Population: 6 DiverFish,1 Snail
Current Tank temperature: 12 C
Amount of food in tank  : 0 units
```