P02 Fish Tank 1000

Overview

This assignment involves developing a graphical application to simulate the behavior of a set of fish while swimming in a fish tank. We are going to continue adding features to this program in p04 and p05. So, be sure to keep your code clean, clear, and well commented.

For this week, we'll focus on developing an application named Fish Tank 1000 that allows the user to add up to 8 fish to a fish tank, and move them arbitrary within the display window using the mouse. No specific action or behavior will be defined for these fish at this stage. The Graphical User Interface (GUI) application will be written using a provided PApplet object and PImage objects defined within the processing library. Fig. 1 shows an example of what this program might look like when it is done.

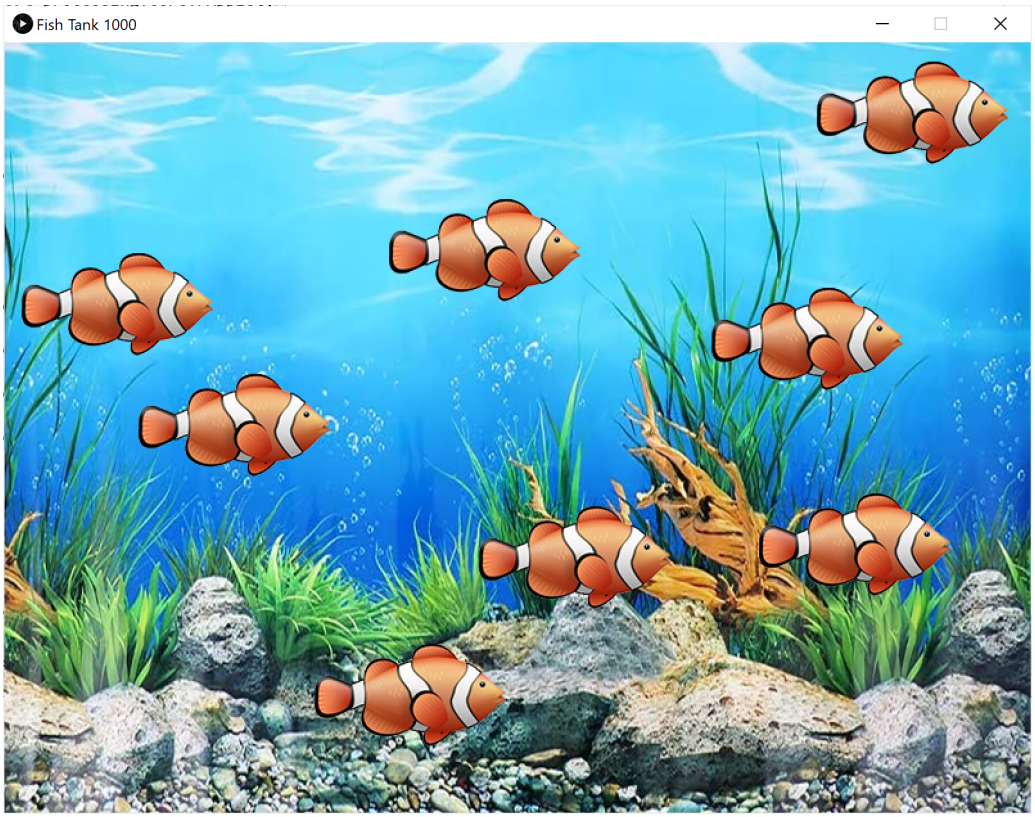


Figure 1: Fish Tank 1000 Display Window

Learning Objectives

The goals of this assignment include:

- Practice how to create and use objects,
- Develop the basis for creating an interactive graphical application,
- Give you experience working with callback methods to define how your program responds to mouse-based input or key-pressed.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P02 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 09/19/2021**. Access to the pre-assignment quiz will be unavailable passing its deadline.
25 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
20 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.

Assignment Requirements and Notes

- Pair programming is **NOT ALLOWED** for this assignment. You MUST complete and submit your p02 individually.
- The ONLY import statements that you may include in this assignment are:

```
import java.util.Random;
import java.io.File;
import processing.core.PApplet;
import processing.core.PImage;
```

- This may be your first experience with developing a graphic application using the java processing library. Make sure to read carefully through the specification provided in this

write-up. This assignment requires clear understanding of its instructions. Read TWICE the instructions and do not hesitate to ask for clarification on piazza if you find any ambiguity.

- The automated grading tests in gradescope are **NOT using the full processing library** when grading your code. They only know about the following fields and methods (referenced directly from this assignment). If you are using any other fields, methods, or classes from the processing library, this will cause problems for the automated grading tests. Such references must be replaced with references to one or more of the following:
  - **two** fields from `PImage`: int `width`, and int `height`.
  - **three** fields from `PApplet`: int `mouseX`, int `mouseY`, and boolean `mousePressed`.
  - **three** methods from `PApplet`: PImage `loadImage(String)`, void `image(PImage,int,int)`, and void `main(String)`.
- You MUST NOT add any additional fields either instance or static to your program, and any public methods either static or instance to your program, other than those defined in this write-up.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private static methods to help implement the different public static methods defined in this program, if needed.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- **NONE** of the callback methods `(setup(), draw(), mousePressed(), mouseReleased(), and keyPressed()` defined later in this write-up should be called explicitly in this program.
- There is NO tester class to be implemented for this assignment. You can print some messages to the console to help you debugging some methods.
- All the methods in this programming assignment MUST be static. We are focusing on procedural programming through p04.
- You HAVE TO adhere to the Academic Conduct Expectations and Advice

1 GETTING STARTED

To get started, let's first create a new Java11 project within Eclipse. You can name this project whatever you like, but "P02 Fish Tank 1000" is a descriptive choice. Then, create a

new class named `FishTank` with a public static void main(String[] args) method stub. This class represents the main class in your program. This process is described near the end of the Eclipse installation instructions here. DO NOT include a package statement at the top of your `FishTank` class (leave it in the default package).

1.1 Download p2core.jar file and add it to your project build path

We have prepared a jar file that contains the processing library, along with a few extra object types to help you build this and future assignments. Download this p2core.jar file and copy it into the project folder that you just created. Then, right-click on this file in the "Package Explorer" within Eclipse, choose "Build Path" and then "Add to Build Path" from the menu. Note: If the .jar file is not immediately visible within Eclipse's Package Explorer, try right-clicking on your project folder and selecting "Refresh".

(**Note that** for Chrome users on MAC, Chrome may block the the jar file and incorrectly reports it as a malicious file. To be able to copy the downloaded jar file, Go to "chrome://downloads/" and click on "Show in folder" to open the folder where your jar file is located.)

If the "Build Path" entry is missing when you right click on the jar file in the "Package Explorer", follow the next set of instructions to add the jar file to the build path:

1. Right-click on the project and choose "Properties".
2. Click on the "Java Build Path" option in the left side menu.
3. From the Java Build Path window, click on the "Libraries" Tab.
4. You can add the "p2core.jar" file located in your project folder by clicking "Add JARs..." from the right side menu.
5. Click on "Apply" button.

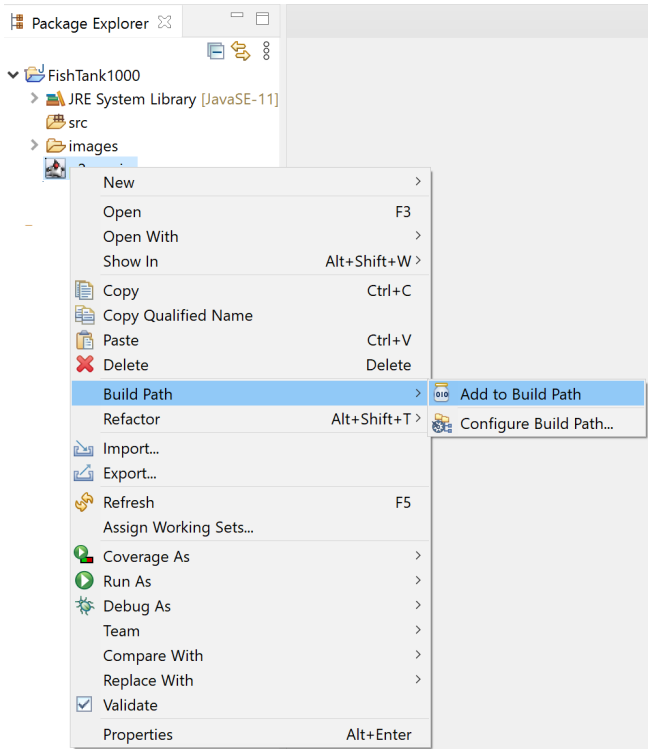
This operation is illustrated in Fig. 2 (a).

1.2 Check your project setup

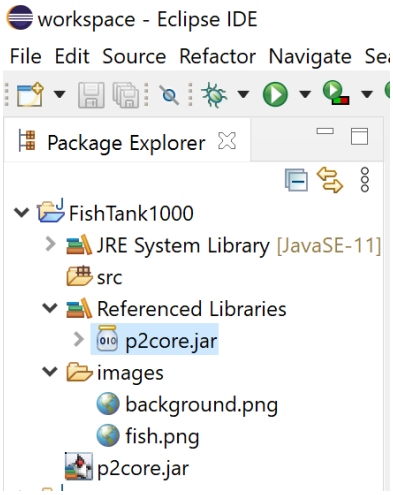
Now, to test that the p2core.jar file library is added appropriately to the build path of your project, try running your program with the following method being called from main() method.

```
Utility.startApplication(); // starts the application
```

If everything is working correctly, you should see a blank window that appears with the title, "Fish Tank 1000" as depicted in Fig. 3. `Please consult piazza or one of the consultants, if you have any problems with this setup before proceeding.`



(a) Adding p2core.jar to build path



(b) P02 package explorer

Figure 2: Fish Tank Application - Getting Started

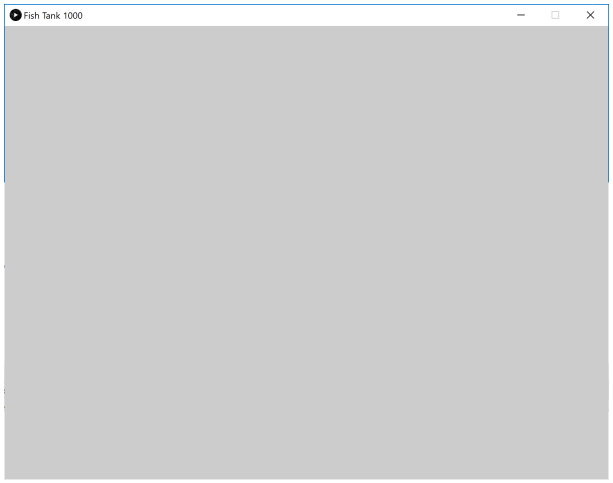


Figure 3: Fish Tank 1000 - Blank Screen Window

Note that the startApplication() method from the provided Utility class, provided in the p2core jar file, creates the main window for the application, and then repeatedly updates its appearance and checks for user input. It also checks if specific callback methods have been defined in the `FishTank` class. Callback methods specify additional computation that should happen when the program begins, the user pressed a key or a mouse button, and every time the display window is repeatedly redrawn to the screen.

1.3 Download images for your Fish Tank application

Create a folder called images within your project folder in Eclipse: this can be done by right clicking your project in the "Project Explorer" and selecting $New > Folder$. Then, download these two images and copy them into your new "images" folder: background.png, and fish.png by following the links, then right-clicking on the images, and selecting "Save Image as...". If you don't see them in Eclipse immediately, try right-clicking on your project folder and selecting "Refresh".

The organization of your p02 project through eclipse's package explorer is illustrated by Fig. 2 (b).

1.4 Overview of the class Fish

The Fish class represents the data type for fish objects that will be created and used in our `FishTank` application. The javadoc documentation for this class is provided here. Make sure to read the description of the constructor and the different methods implemented in the Fish class carefully. You do not need to implement this class or any of its declared methods. The class Fish is entirely provided for you in the p2core.jar file. Its implementation details are hidden for you as users of that class. All the important information required to use its public methods appropriately are provided in these javadocs.

1.5 Overview of Callback Methods

Note that the Utility.startApplication() creates the display window, sets its dimension, and checks for callback methods. A callback method specifies additional computation that should happen in response to specific events. In this assignment, we are going to implement the following callback methods.

- `setup()` method: This method is automatically called by Utility.startApplication() when the program begins. It creates and initializes the different data fields defined in your program, and configures the different graphical settings of your application, such as loading the background image, setting the dimensions of the display window, etc.
- `draw()` method: This method continuously executes the lines of code contained inside its block until the program is stopped. It draws repeatedly the display window to the screen. All processing programs update the screen at the end of the draw() method.
- `mousePressed()`: The block of code defined in this method is automatically executed each time the mouse bottom is pressed.
- `mouseReleased()`: The block of code defined in this method is automatically executed each time the mouse bottom is released.

- `keyPressed()`: The block of code defined in this method is automatically executed each time a key is pressed.

Note that NONE of the above callback method should be called explicitly in your program. They are automatically called by every processing programs in response to specific events as described above.

2 Visualizing the Fish Tank Display Window

2.1 Declare `FishTank` Static Fields

Add the following static fields (class variables) to your `FishTank` class. Put them outside of any method including, the main(). The top of the class body is be a good placement where to declare them.

```
private static PApplet processing; // PApplet object that represents the graphic
                                   // interface of the JunglePark application
private static PImage backgroundImage; // PImage object that represents the
                                       // background image
private static Fish[] fishes; // perfect size array storing the different fish present
                              // in the fish tank. These fish can be of different species.
private static Random randGen; // Generator of random numbers
```

To avoid compile without errors, check that `import java.util.Random`, `processing.core.PApplet`, and `processing.core.PImage` classes are already imported at the top of your FishTank.java source file.

Recall that you should not add any other variable (instance or static data field) to the FishTank class. You can define local variables that you may need to implement the methods that will be defined in this program.

2.2 Define the setup callback method

Recall that `Utility.startApplication()` creates the display window, sets its dimension, and checks for callback methods. The first callback method that we are going to implement in this program is the **setup()** method. You should define this method in your `FishTank` class with **EXACTLY** the following signature:

```
/**
 * Defines the initial environment properties of this application
 * @param processingObj a reference to the graphic display window of this application
 */
public static void setup(PApplet processingObj) {}
```

The **setup()** method receives an argument of type PApplet that is passed to it from the Utility class. It is used to define initial environment properties such as screen size and to load background images and fonts as the program starts. **There can only be one setup() method in the whole program and it is run automatically once when the program starts**.

To convince yourself that this method is being called once when your program starts up, try adding a print statement to the `setup()` method, then run your program. Note that the output of your System.out.println() print statement will be displayed in the console and not on the graphic display window. After this test, you can remove the print statement before proceeding. Now, follow the next steps to visualize the fish tank window as it is shown in Fig.4.

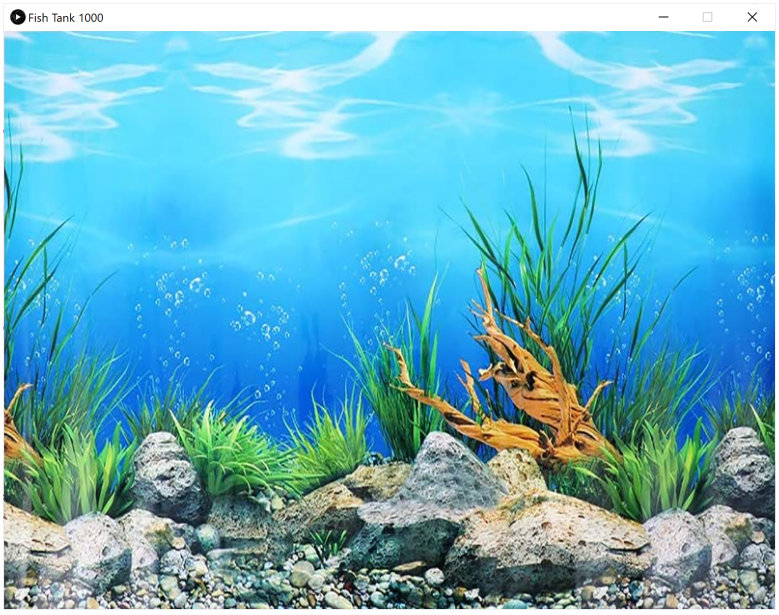


Figure 4: Application Display Window with Background

2.3 Draw the background image at the top of the display window

We defined a processing variable of type PApplet. This variable refers to an object that contains most of the drawing functionality from the processing library. Since this processing object is one that you cannot create on your own at this point of progress of the course, the `Utility.startApplication()` method, will pass it as parameter into the setup() method.

To draw a background image at the top of the display window, follow the next instructions.

1. In the setup() method, set the processing static variable to the one passed as input parameter.

2. Make sure that you have a folder called "images" in your "Package Explorer" and that it

contains two images background.png and fish.png. If it is not the case, go up to "Download images for your Fish Tank application" subsection in the Getting Started section.

3. In the `setup()` callback method, load the background image and store its reference into the backgroundImage variable of type PImage by calling the method loadImage() defined in the processing library as follows.

```
// load the image of the background
backgroundImage = processing.loadImage("images/background.png");
```

4. If you run now your program, you can notice that even though the backgroundImage is loaded in the setup() method, the background image is not yet drawn at the center of the screen of our application. To do so, we need to call the image() method defined in the processing library. The PApplet.image() method draws a PImage object at a given position of the screen. Please find below an example of code that draws an image at the center of our Fish Tank display window. Notice the importance of running this code after calling the background() method, instead of before.

```
// Draw the background image at the center of the screen
processing.image(backgroundImage, processing.width / 2, processing.height / 2);
// width [resp. height]: System variable of the processing library that stores
// the width [resp. height] of the display window.
```

The processing library defines the PImage class as a datatype used for storing images. A PImage object is used to represent an image that can be loaded from a file by calling loadImage() method, and then drawn to the screen at a given position by calling image() method. Processing can display .gif, .jpg, .tga, and .png images.

2.4 Draw a fish image at the top of the background

You can now load and draw the fish image to the center of the screen in a similar way. You can also change the position where the fish's image can be drawn to the screen. After that, leave only the code that loads and draws the background image on the top of the display window and remove the lines that draws the fish's image to the screen.

3 Adding Fish to the Fish Tank

3.1 Add one Fish object and draw it to the center of the screen

Let's now create one fish object and draw it to the center of the screen. Recall that we defined the static variable ***fishes*** as a reference to an array where Fish objects present in our Fish

Tank will be stored.

In the `setup()` method, create (using the new keyword) the array *fishes* which can store up to 8 Fish references. This array must begin with 8 null references. Note that the array *fishes* **DOES NOT** refer to an oversize array. It is defined by its reference ONLY. This means that its size equals its length by default.

Now, create one Fish object and store its reference at the element of index 0 in the *fishes* array. Recall that every Fish object is created by calling one constructor of the class `Fish` using the keyword **new**. The Fish class was provided for you in the jar file. Do not create this class. To have an idea of the different constructors and methods defined in the Fish class, please refer to the details provided in these javadocs.

The following statement of code creates a new Fish object and locates it at the center of the display window defined by the reference processing of type PApplet, then stores its reference at the first position of the *fishes* array.

```
fishes[0] = new Fish(processing);
```

Note the importance to add the above statement after the *fishes* array is created. Otherwise, a NullPointerException will be thrown.

You can also use the second constructor provided in the Fish class as follows.

```
fishes[0] = new Fish(processing, processing.width / 2, processing.height / 2);
```

Now, to draw the Fish's image to the application display window, call the Fish object's *draw()* method. An example of invoking the Fish object's *draw()* method is provided in the following line of code.

```
fishes[i].draw(); // where i is the index of the created Fish in the fishes array.
```

Try now to run your program with only one Fish object created and added to the Fish Tank at the center of the screen. You should have the output illustrated by Fig.5.

3.2 Add fishes and draw them at random positions of the screen

Now, we would like to add up to 8 fishes to our fish tank and draw them at random positions of the window display. To generate these random positions, we are going to use the *randGen* static variable of type Random that we already defined outside of any method. First, make sure to create a Random object in the *setup()* method and **assign its reference to the static variable** *randGen*.

The following demonstrates an example of use of a Random object to create random x and y positions in the display window.

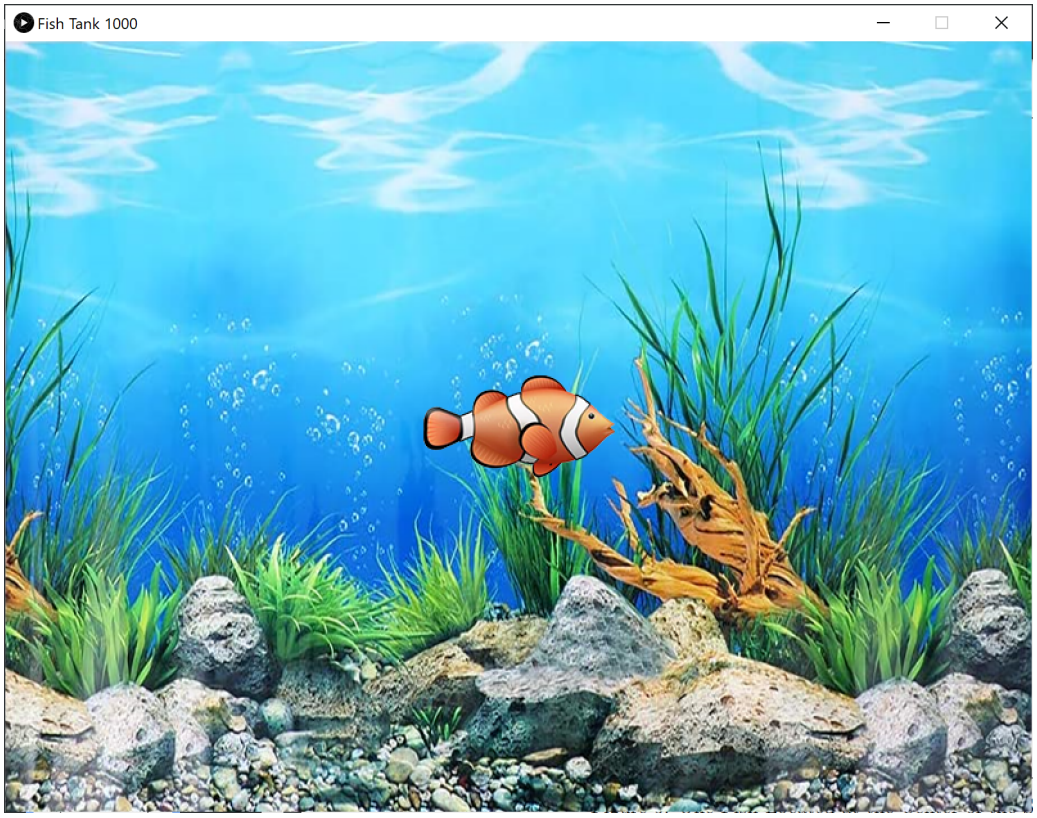


Figure 5: One Fish Located at the Center of the Screen

```
(float)randGen.nextInt(processing.width) // generates a random x-position of type
                                         // float within the width of the display window
(float)randGen.nextInt(processing.height) // generates a random y-position of type
                                          // float within the height of the display window
```

To create a Fish at a given position of the display window, call the constructor of the class Fish which takes three input parameters `Fish(PApplet, float, float)`. Then, you can store the reference of the created Fish object returned by the Fish constructor call into the position 0 of the array *fishes*.

Next, you can add a `for-loop` to draw the Fish objects pointed by each of the `non-null` references in the *fishes* array by calling each Fish object's draw() method appropriately.

Having this done, you can now test your program with different numbers of fishes at random positions within the display window, and with a *fishes* array of different sizes.

Before moving on to the next step, make *fishes* reference an array of length eight. **Make all but the first of these references null, and make the first one a reference to a Fish object located at a random position of the screen.**

4 Moving fish in the Fish Tank

4.1 Define the draw() callback method

Since all of our code is currently in the `setup()` method, it is only being run once when our program begins. To be able to move a Fish around, we'll need to repeatedly draw it at different positions at different times. We'll do this through another callback method called `draw()`. The draw() method must have the following signature.

```
/**
 * Draws and updates the application display window.
 * This callback method called in an infinite loop.
 */
public static void draw() {}
```

- Note that `draw()` method will continuously draw the application display window and updates its content with respect to any change or any event which affects its appearance. Try printing a test message within this method's definition, to convince yourself that this method is being called repeatedly by the Utility class as a result of your Utility.startApplication() call. The text message will be repeatedly printed to the console. After this test, you can remove the print statement.

- Now, move the statement (for-loop) that draws the Fish objects stored in the *fishes* array from setup() into this draw() method.

- To give the user control over the position of the Fish object stored at fishes[0], we are going to set the position of this Fish to the position of the mouse in the draw() method just before drawing its image. To do so, call the `setPositionX()` and `setPositionY()` methods defined in the Fish class as documented in these javadocs.

- Note also that you can access the mouse position through the fields **mouseX** and **mouseY** using the *processing* reference. The mouseX and mouseY are public instance fields which belong the processing PApplet object. They always contain the current horizontal and vertical position coordinates of the mouse.

- Having this done, you'll notice that the Fish always follows the mouse, and that it simultaneously appears in its current position along with all of its previous locations. To prevent it from looking like there are multiple fishes, move the code that draws the background image from the setup() method to the beginning of your draw() method. Keep the line of code which loads the background image in the setup() method. The background image should only be loaded once per call of the setup() method, and it should be drawn to the screen per each call of the draw() method.

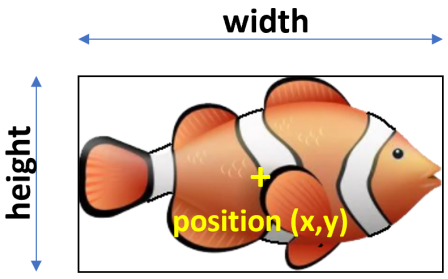
4.2 Move the fish only when it is being dragged

Now, we only want the Fish to move, after the user has clicked on the Fish image. To check if a Fish is being dragged by the user, define a method called `isMouseOver()` with the exact following signature and add it to your `FishTank` class. The isMouseOver() method should return true if the mouse is over the image of the Fish object passed as input parameter, and false otherwise.

```
/**
 * Checks if the mouse is over a specific Fish whose reference is provided
 * as input parameter
 *
 * @param Fish reference to a specific fish
 * @return true if the mouse is over the specific Fish object (i.e. over
 *         the image of the Fish), false otherwise
 */
public static boolean isMouseOver(Fish fish) {}
```

To determine whether the mouse is over the fish passed as input to the `isMouseOver()` method, use `width` and `height` public fields defined within the image (of type PImage) of the Fish object.

- Read carefully through the methods defined in these javadocs to determine which method returns a reference to the Fish's image of type PImage.
- Every PImage object has two attributes (public fields) `.width` and `.height` accessible through its reference.
- Note that the center of the image is the position of the Fish object within the display window (fish.getPositionX(), fish.getPositionY()) as demonstrated in the following figure.



- Given this done, remove now the lines of code which set the positionX and positionY of the Fish object to the mouse positions from the draw() method. Do not remove your calls to the Fish objects' draw() method.

Let's now define `mousePressed()` and `mouseReleased()` callback methods. Their signatures must be as follows.

```
/**
 * Callback method called each time the user presses the mouse
 */
public static void mousePressed() {}

/**
 * Callback method called each time the mouse is released
 */
public static void mouseReleased() {}
```

Note that our Utility class will automatically call mousePressed() each time the mouse button is pressed down. Then, the mouseReleased() method will be called once when the mouse button is later released.

Now, implement the `mousePressed()` method. This method should check if the mouse is over one of the Fish objects stored in the *fishes* array and start dragging it. To do so, you can call the `setDragging()` method defined in the Fish class (see these javadocs for details). If the mouse is over more than one Fish, only the Fish stored at the lowest index within the *fishes* array will be dragged. Note also that when dragging a Fish, the other fishes have to remain visible in the fish tank display window.

Next, implement the `mouseReleased()` method. No Fish must be dragged when the mouse is released. To do so, you have to setDragging() every Fish stored in the fishes array to false. Recall that the *fishes* object is not an oversize array. It can contain a null reference at any index position.

With all of this in place, the user should be able to drag and drop up to eight fish within the fish tank area. If a Fish is being dragged, the draw() method defined in the Fish class sets the position of that Fish to the position of the mouse and draws the Fish's image to the screen window accordingly.

Recall that `draw()` method will redraw our display window each time an event registered occurs, *such as the mouse is pressed down or released or dragged or moved or a key is pressed.*

5 Adding fishes to the Fish Tank by Pressing F-key

Now, rather than creating the Fish objects in the setup() method, let's start with eight null values in the *Fishes* array. At this level, remove any line of code which creates a new Fish object from your setup() or draw() methods. We'll allow the user to add new fishes (up to eight fish) into the fish tank by pressing the F-key. Whenever this is 'f' or 'F' for Fish, search through the fishes array for a null reference, if found replace the first (lowest index) null reference with a new Fish object located at a random position of the display window. This behavior must

be implemented in the callback method `keyPressed()` which must have the EXACT following signature.

```
/**
 * Callback method called each time the user presses a key
 */
  public static void keyPressed() {}
```

Note that each time the user presses any key, the `keyPressed()` callback method will be executed. You can check which key was pressed using the public "***key***" field which belongs to the processing object.

6 Removing fishes from the tank by Pressing R-key

Finally, we would like to allow the user to remove fishes from the fish tank. Update the callback `keyPressed()` method such that if the **'R'** or **'r' key** is pressed while the mouse is over a Fish, that Fish will be removed from the tank (meaning that its reference will be set to null) until another Fish is created in its place (by pressing the F-key). DO NOT move the null references to the end of the array *fishes*, or compact its contents.

If the mouse is over multiple fishes while the R-key is pressed (meaning in case of overlapping fishes), the Fish stored at the lowest index position within the *fishes* array will be removed.

7 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only ONE file that you must submit is `FishTank.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of **9:59PM on September 23**rd. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline.