Assignment4 / NS Shaft+/

ASSIGNMENT 4

NS Shaft+

// Players control a character that continuously falls down platforms in a vertical shaft.The goal is to land on safe platforms while avoiding traps or falling off-screen.

The player is a fresh shrimp.
You need to survive for 60 seconds — once the timer hits 60, the fryer appears, and you can jump into it to transform into a delicious piece of tempura shrimp!

What you'll learn...?

Inheritance

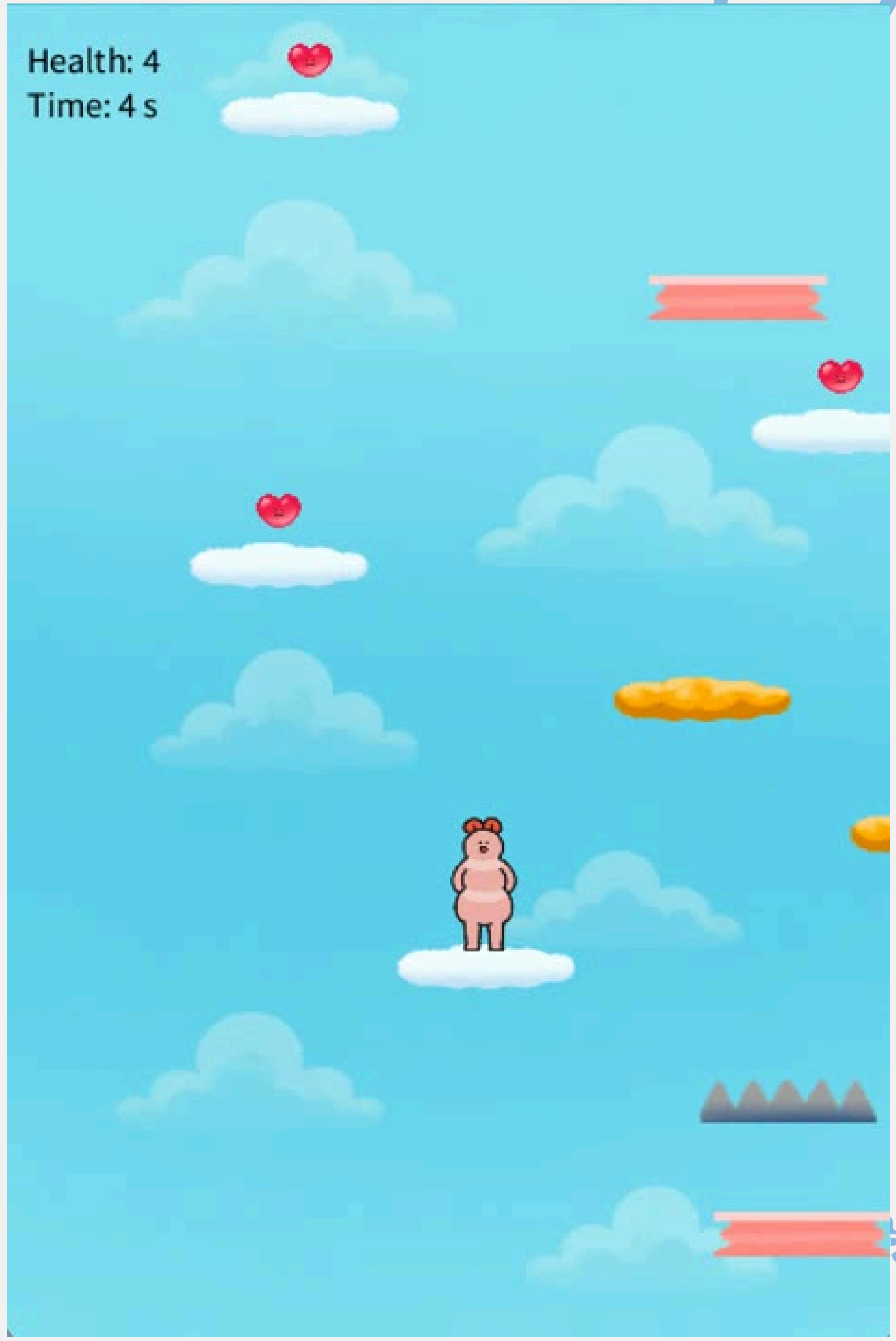
Break your code into small functions like scrollBackground() and movePlayer() to keep draw() clean and readable.

Override

Turn the player and platform into classes with their own update() and display() methods for better structure.

super()

Use an array of Platform objects to manage multiple platforms efficiently with a simple loop.



BREAKDOWN(拆解)

Practice 1	finish **SpikyPlatform**
Practice 2	finish **FragilePlatform**
Practice 3	finish **HealPlatform**
Practice 4	add **sound effects** to each Platforms
Extra	Make a cool ending for the game!

Inheritance

```
class XXX  extends XX {
```

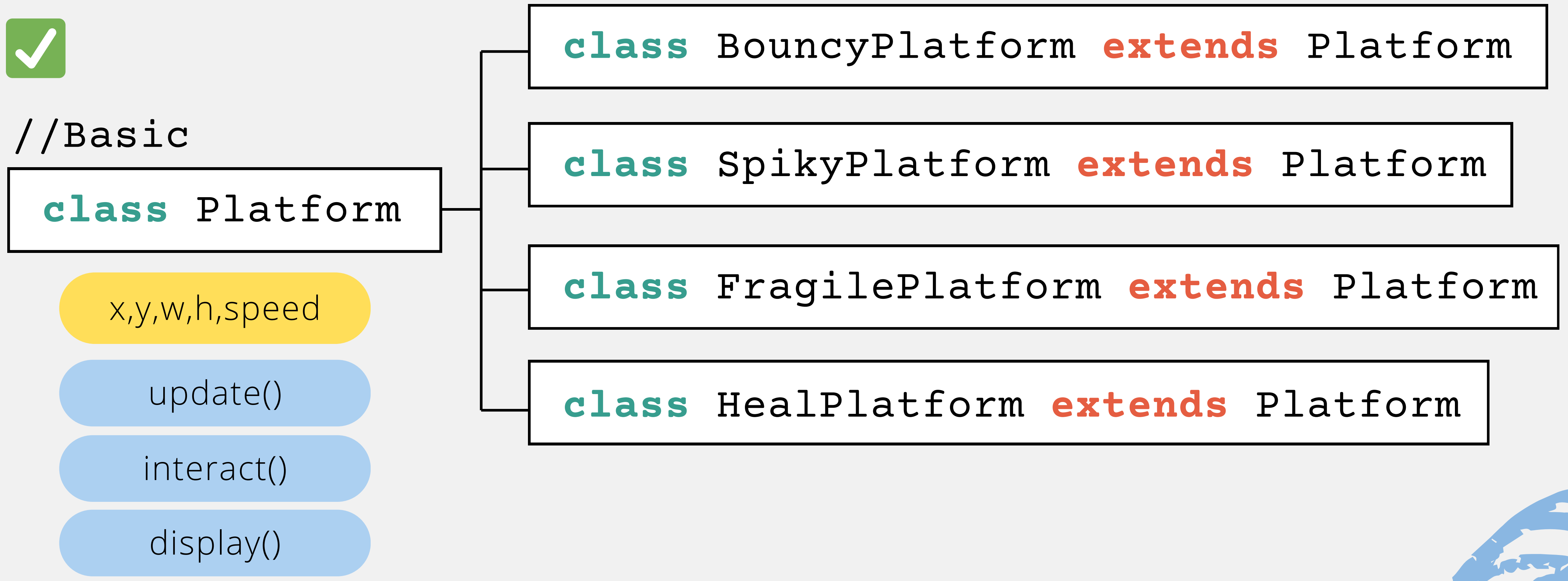
💡 **Inheritance** lets one class (child class / superclass) reuse the properties and methods of another class (parent class / subclass).

- Avoid code duplication
- Reuse shared logic
- Extend or customize behavior
- Organize code into logical layers

🚫

```
class Platform
class BouncyPlatform
class SpikyPlatform
class FragilePlatform
class HealPlatform
```

✅



```
//Basic
class Platform
```

```
class BouncyPlatform extends Platform
class SpikyPlatform extends Platform
class FragilePlatform extends Platform
class HealPlatform extends Platform
```

override

Replaces a method from the **parent class** with a new version in the child class.

💡 **Overriding** means that a subclass **redefines a method** that already exists in its superclass — using the **same method name and parameters**, but changing what it does.

✅ When to use?

>>When we inherit a method but need to **customize** the behavior.

📦 Example:

```
class Platform {
  void display() {
    println("Displaying a normal platform");
  }
}
```

```
class BouncyPlatform {
  void display() {
    println("Displaying a bouncy platform");
  }
}
```

🔎 Key Points:

- The **method name** must match exactly.
- If you override a method and still want to keep some of the original behavior...? →

super()

Calls the **parent class's constructor or method** in a child class.

💡 **super()** is used to refer to the parent class.
It allows the child class to **reuse code from its parent class**.

🔧 Two main uses:

1. Calling the parent class constructor (建構子)

Use this when you want to initialize values from the parent class.

```
super(parameters);
```

ex. `super(x,y);`

2. Calling a method from the parent class

Use this when you've overridden a method
but still want to run the original version.

```
super.methodName();
```

ex. `super.display();`

Summary

inheritance / override / super()

override	Customize specific methods in the subclass
Inheritance	Share common code between classes
super()	Reuse parts of the parent class

Example

Create BouncyPlatform by inheriting class Platform

```
class Platform {
  float x, y, w = 80, h = 20, speed = 2; // Position, size, and speed of the platform
  boolean recycleFlag = false; // Flag to indicate if the platform needs recycling

  Platform(float tempX, float tempY) {
    x = tempX;
    y = tempY;
  }

  void update() {
    y -= speed; // Move the platform up
    if (y < -h) {
      recycleFlag = true; // Mark the platform for recycling
    }
  }

  void interact(Player player) {
    player.ySpeed = 0; // Reset the player's ySpeed
    player.y = y - player.h + player.feetOffset; // Place the player on top of the platform
    player.y -= speed; // Move the player up with the platform
  }

  void display() {
    image(platformImage, x, y, w, h); // Draw the platform
  }
}
```

Basic position

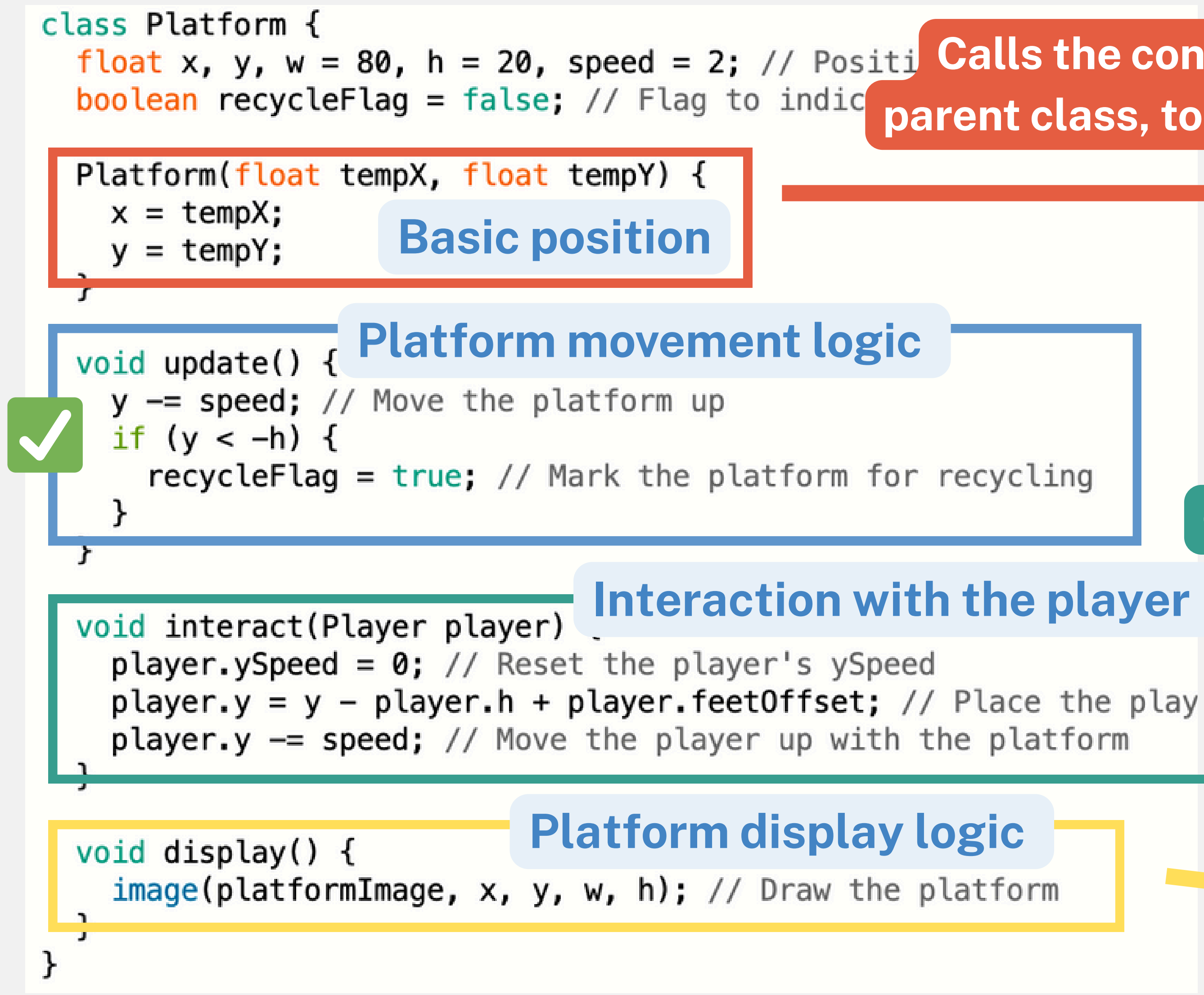
Platform movement logic

Interaction with the player

Platform display logic

※class Platform

※class Platform



```
class Platform {
  float x, y, w = 80, h = 20, speed = 2; // Positi
  boolean recycleFlag = false; // Flag to indic

  Platform(float tempX, float tempY) {
    x = tempX;
    y = tempY;
  }

  void update() {
    y -= speed; // Move the platform up
    if (y < -h) {
      recycleFlag = true; // Mark the platform for recycling
    }
  }

  void interact(Player player) {
    player.ySpeed = 0; // Reset the player's ySpeed
    player.y = y - player.h + player.feetOffset; // Place the play
    player.y -= speed; // Move the player up with the platform
  }

  void display() {
    image(platformImage, x, y, w, h); // Draw the platform
  }
}
```

Basic position

Platform movement logic

Interaction with the player

Platform display logic

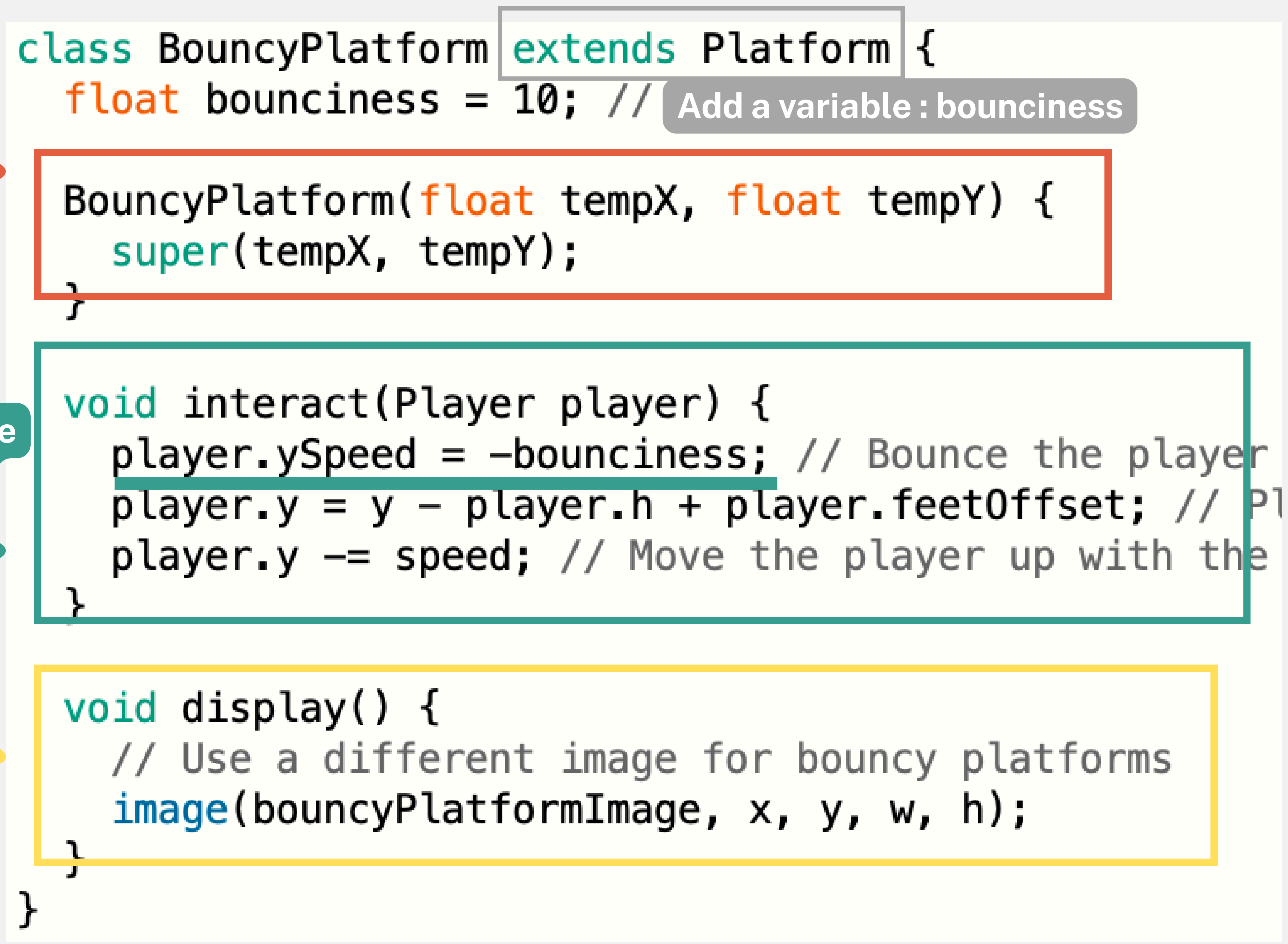
Calls the constructor of the parent class, to initialize x and y.

Override the first line and keep the rest

Overridden

cloud.png

※class BouncyPlatform

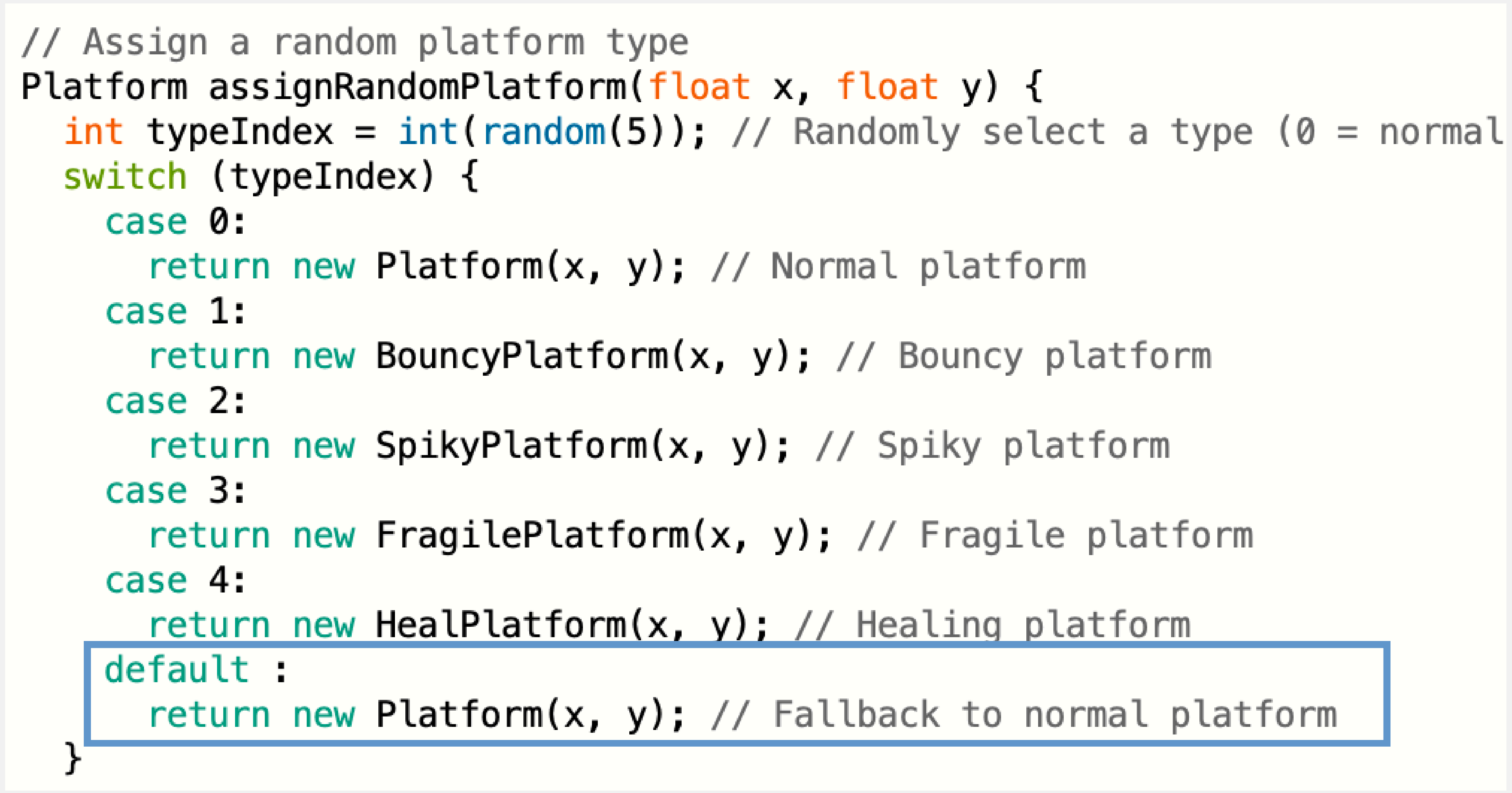


```
class BouncyPlatform extends Platform {
  float bounciness = 10; //

  BouncyPlatform(float tempX, float tempY) {
    super(tempX, tempY);
  }

  void interact(Player player) {
    player.ySpeed = -bounciness; // Bounce the player
    player.y = y - player.h + player.feetOffset; // Pl
    player.y -= speed; // Move the player up with the
  }

  void display() {
    // Use a different image for bouncy platforms
    image(bouncyPlatformImage, x, y, w, h);
  }
}
```

Add a variable : bounciness

bouncy_Platform.png

※In the main code, we add these to randomly generate different platform types.



```
// Assign a random platform type
Platform assignRandomPlatform(float x, float y) {
  int typeIndex = int(random(5)); // Randomly select a type (0 = normal
  switch (typeIndex) {
    case 0:
      return new Platform(x, y); // Normal platform
    case 1:
      return new BouncyPlatform(x, y); // Bouncy platform
    case 2:
      return new SpikyPlatform(x, y); // Spiky platform
    case 3:
      return new FragilePlatform(x, y); // Fragile platform
    case 4:
      return new HealPlatform(x, y); // Healing platform
    default :
      return new Platform(x, y); // Fallback to normal platform
  }
```

※When adding a new platform type, you might forget to update the switch statement with a matching case. In that case, the **fallback** ensures a default behavior is still provided.

Click and accept:
https://classroom.github.com/a/Wp1LbDp2

REQUIREMENTS
Stage one ★☆☆

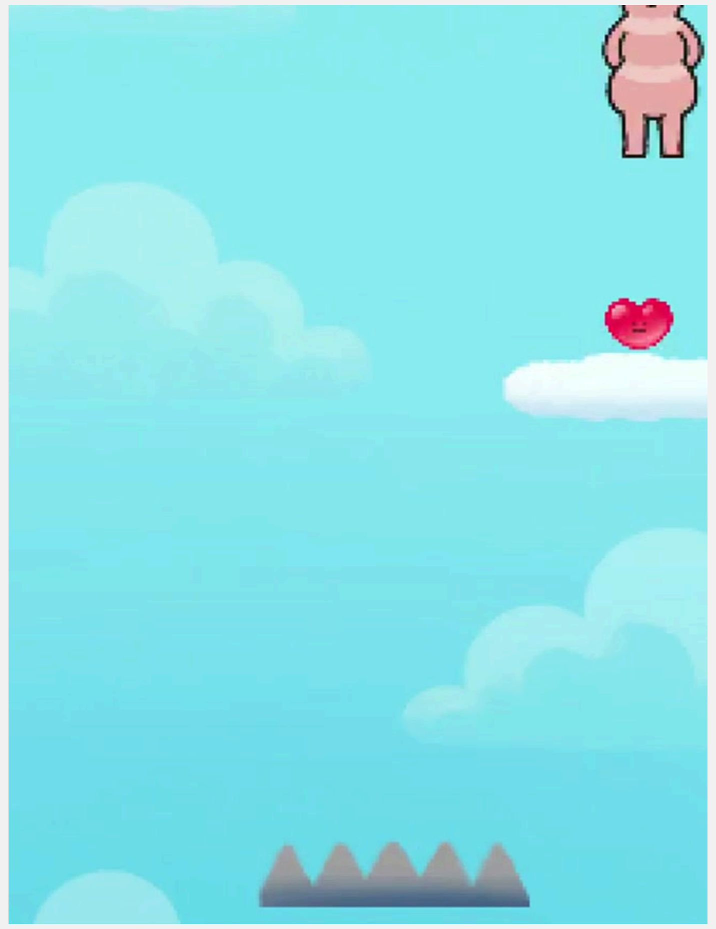
Practice 1

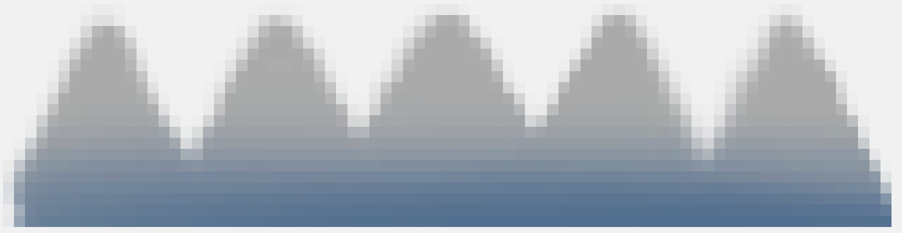
finish **SpikyPlatform**

When the player lands on it:

- It deals damage only once **(use a boolean to check)**
- It still runs the default landing behavior (call parent’s interact()).
- Display a different image and sound for spiky platforms.

```
int damage = 1;
```




spiky_platform.png

REQUIREMENTS

Stage two ★★☆

Practice 2

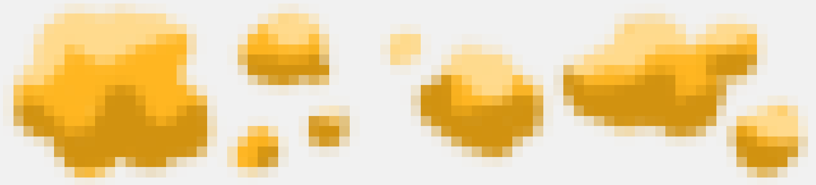
finish **FragilePlatform**

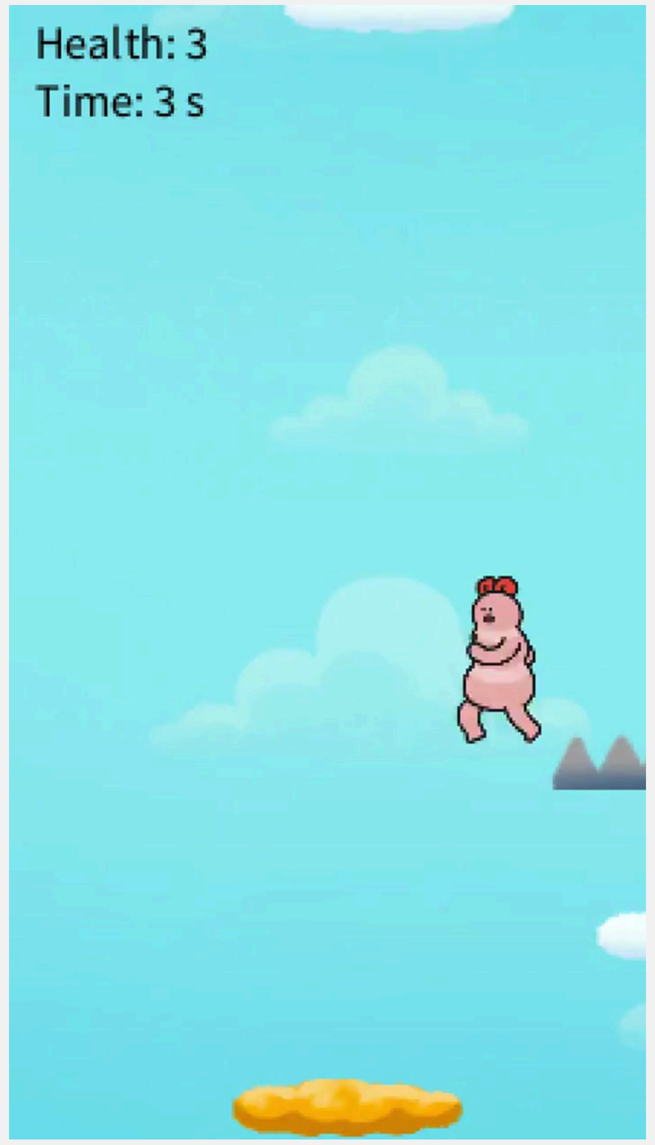
The fragile platform **breaks** after a few seconds.

- The platform should stay solid at first.
- After FRAGILE_PLATFORM_DURATION, the platform breaks and player falls through it. **(Only interact if the platform is not broken)**

```
float duration;
```

fragile_platform.png


fragile_platform_broken.png



REQUIREMENTS
Stage three ★★★

Practice 3 finish **HealPlatform**

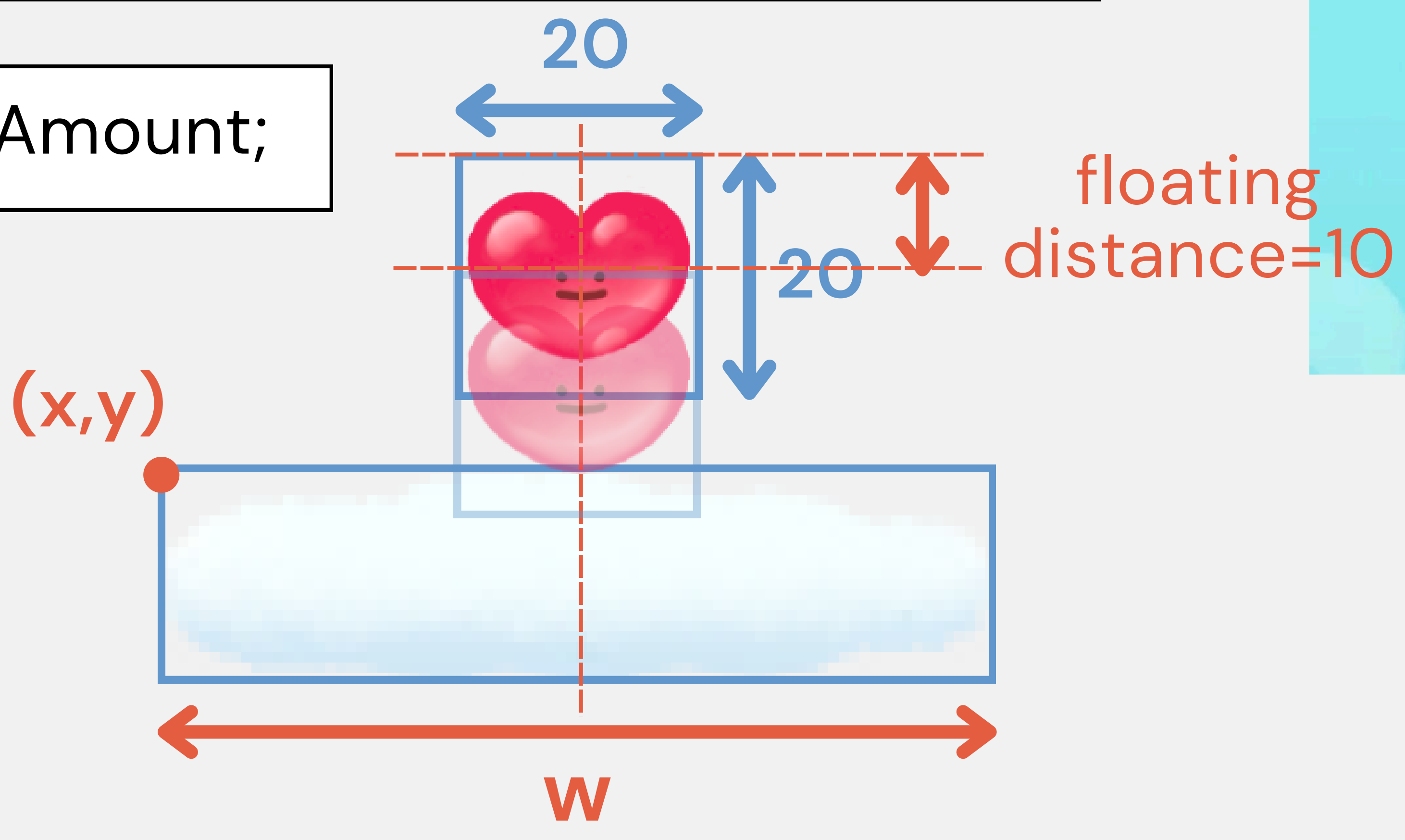
Make a platform with a floating healing potion!

- The potion will be **floating up and down**.
- This platform should heal the player **only once**.
- When the player steps on it, add health and the potion **disappears**.

```
float potionW, potionH;
```

```
int healingAmount;
```

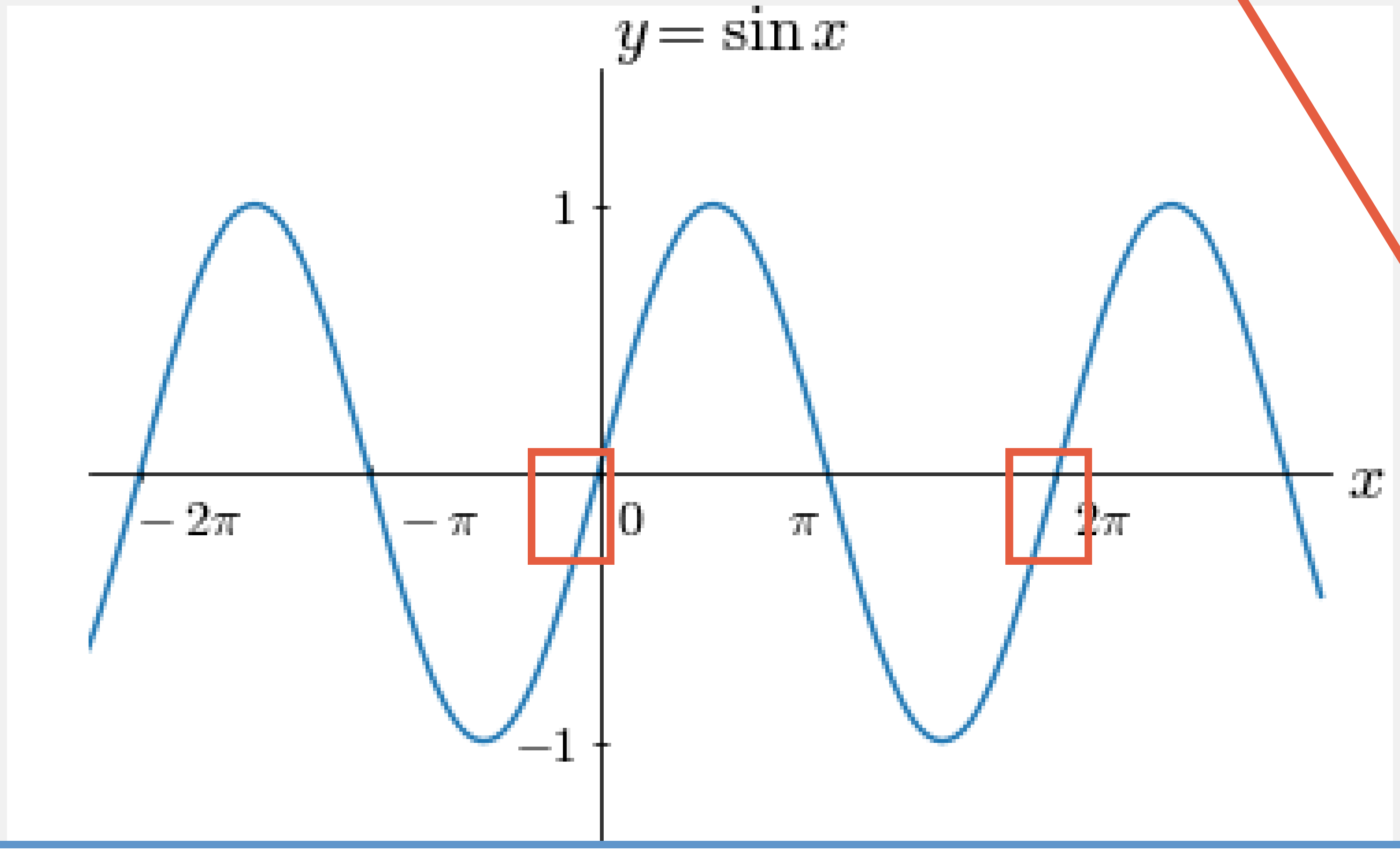
healing_potion.png



Hint

how to make the potion float up and down?

```
float potionY = y - potionH - 5 - 10 * sin(TWO_PI * (frameCount / 60.0));
```



The sin() function creates a smooth wave that goes from -1 to 1.

💡 Using **frameCount** drives the animation over time.

(0÷60=0)
(30÷60=0.5)
(60÷60=1)

💡 Multiplying it by **10** increases the **movement range** (how far it floats)

💡 **-5** shifts the floating center **slightly upward** so the potion appears above the platform.

REQUIREMENTS
Stage four ★★★

Practice 4

add sound effects to each Platforms

When player steps on a platform:

- Play the corresponding sound effect for each platform type.
- **Don't** play the sound while **standing** on the platform.

> To make sure the sound only plays once, use a **boolean** flag to check if the sound has already been triggered.

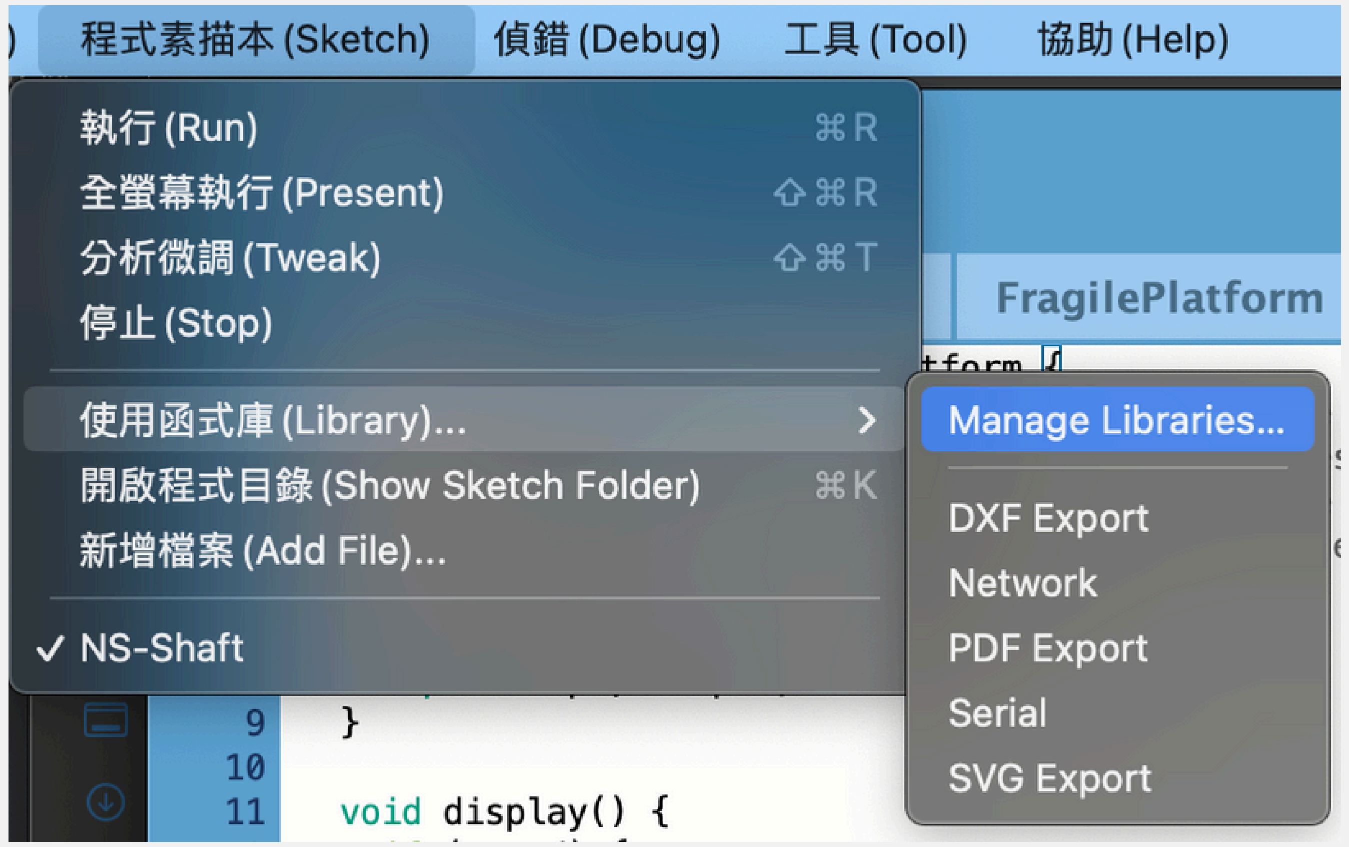
```
boolean playedSound;
```

Play the sound in **interact()**

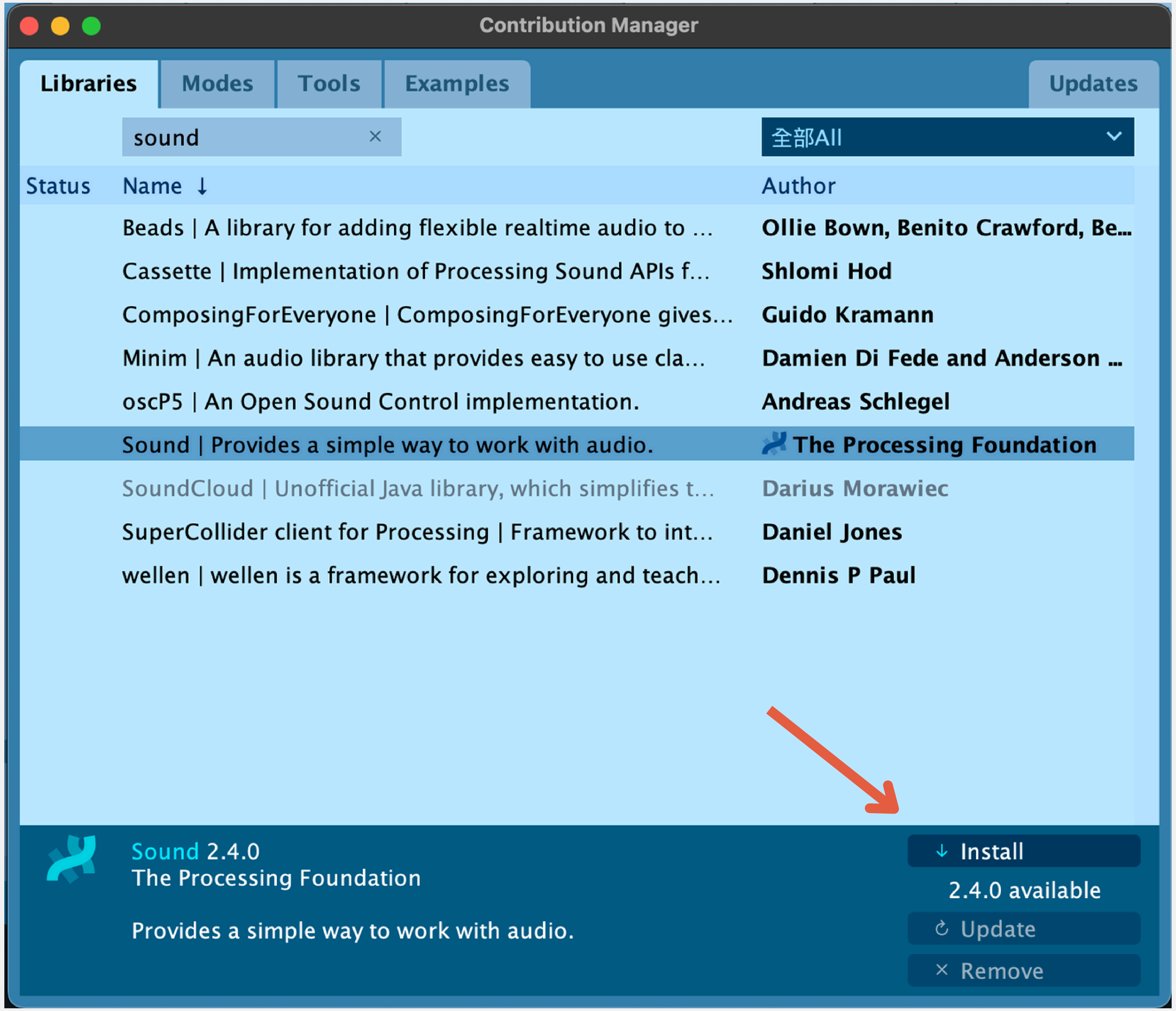
Create a **playPlatformSound()** Function in Platform class

normal.mp3 bouncy.mp3 spiky.mp3 fragile.mp3 fragile_ broken.mp3 heal.mp3

※ Remember to download from the Library first before using SoundFile.



```
}

void display() {
```



Status	Name ↓	Author
	Beads | A library for adding flexible realtime audio to ...	Ollie Bown, Benito Crawford, Be...
	Cassette | Implementation of Processing Sound APIs f...	Shlomi Hod
	ComposingForEveryone | ComposingForEveryone gives...	Guido Kramann
	Minim | An audio library that provides easy to use cla...	Damien Di Fede and Anderson ...
	oscP5 | An Open Sound Control implementation.	Andreas Schlegel
	Sound | Provides a simple way to work with audio.	The Processing Foundation
	SoundCloud | Unofficial Java library, which simplifies t...	Darius Morawiec
	SuperCollider client for Processing | Framework to int...	Daniel Jones
	wellen | wellen is a framework for exploring and teach...	Dennis P Paul

Sound 2.4.0
The Processing Foundation

Provides a simple way to work with audio.

Extra Make a cool ending for the game!

In the **data** folder, there is a PNG image of a **fried shrimp**.

After the player wins the game, the player (a shrimp) should fall into a fryer and turn into a fried shrimp.
>> Use your creativity and this image to **design a interesting game ending screen!**

fried.png

Assignment due: 5/26 12:00pm

remember to come here at 1:20 next week!