

Chapter 5 – Abstraction

Acknowledgement: Michael Kolling, Bruce Chittenden

Why do we need Classes?

- Encapsulation – Information Hiding
- Abstraction - A technique to solve a whole class of problem rather than a specific one.
 - *A more formal Definition - a concept or idea not associated with any specific instance.*

Logical Operators

- Help us create more complex boolean expressions
 - And (&&)
 - Conjunction –true only when both conjuncts are true.
 - Or (||)
 - Disjunction –false only when both disjuncts are false.
 - Not (!)
 - Negation –changes the truth value between false and true.

Complex Boolean Expressions

- Combination of two or more boolean expressions

- If (`!isDown && Greenfoot.isKeyDown("w")`)

```
{
```

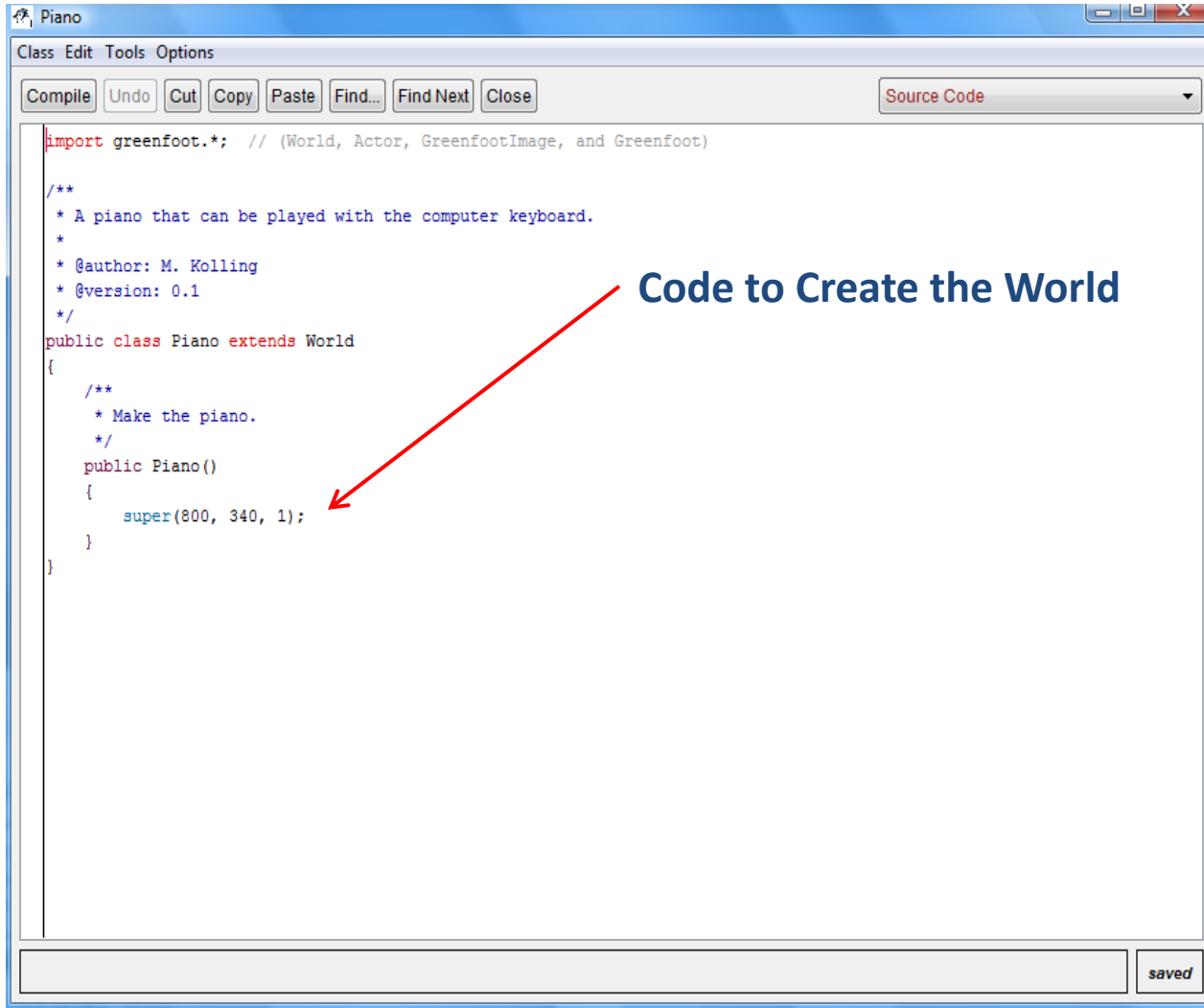
```
    // Set of instructions to execute if the complex expression above  
    evaluates to true.
```

```
}
```

An informal representation of above expression can be

If ((not isDown) and (the keyboard key "w" is down or pressed))
execute the set of instructions

Exercise 5.1



```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A piano that can be played with the computer keyboard.
 *
 * @author: M. Kolling
 * @version: 0.1
 */
public class Piano extends World
{
    /**
     * Make the piano.
     */
    public Piano()
    {
        super(800, 340, 1);
    }
}
```

Code to Create the World

saved

Exercise 5.1

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

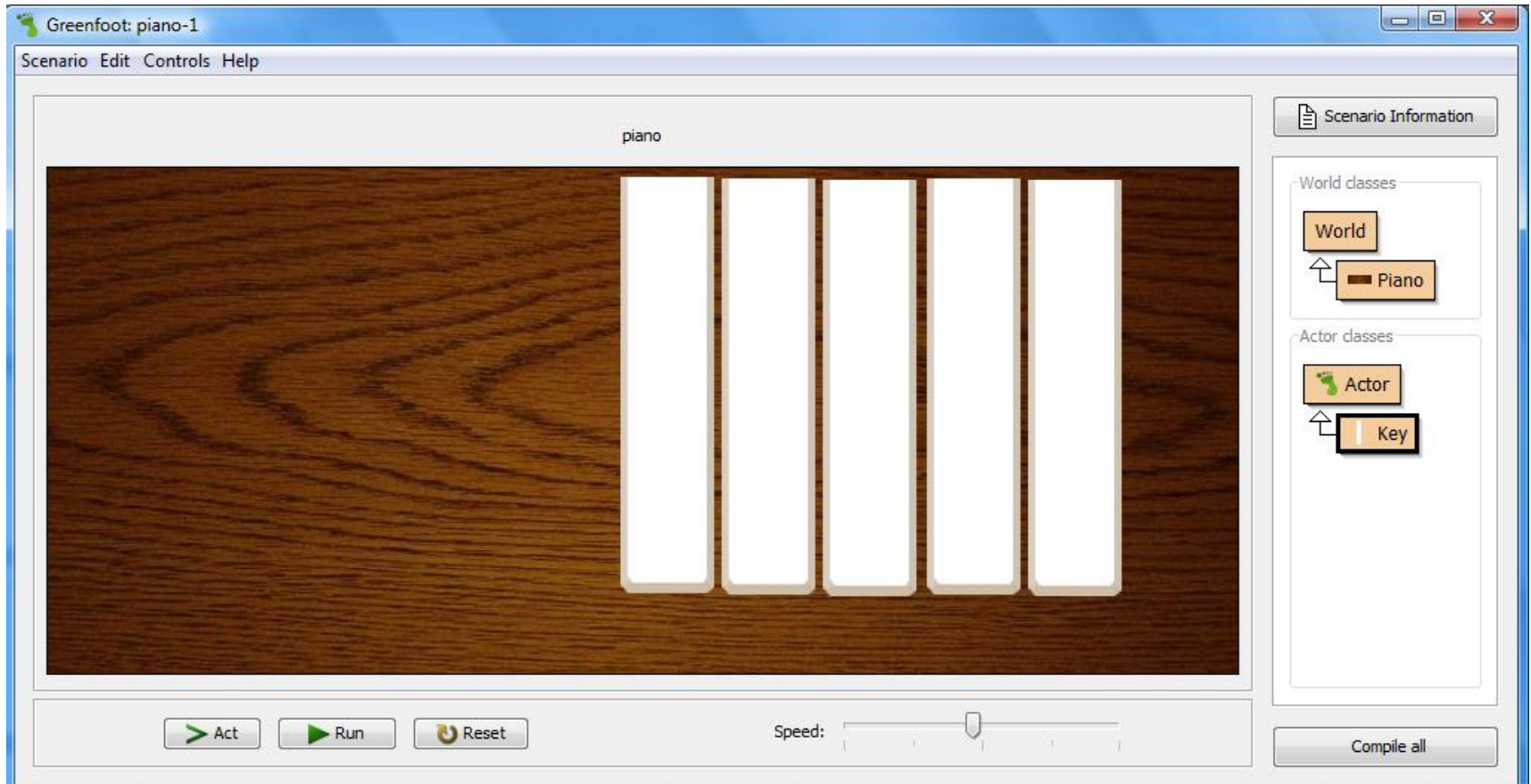
public class Key extends Actor
{
    /**
     * Create a new key.
     */
    public Key()
    {
    }

    /**
     * Do the action for this key.
     */
    public void act()
    {
    }
}
```

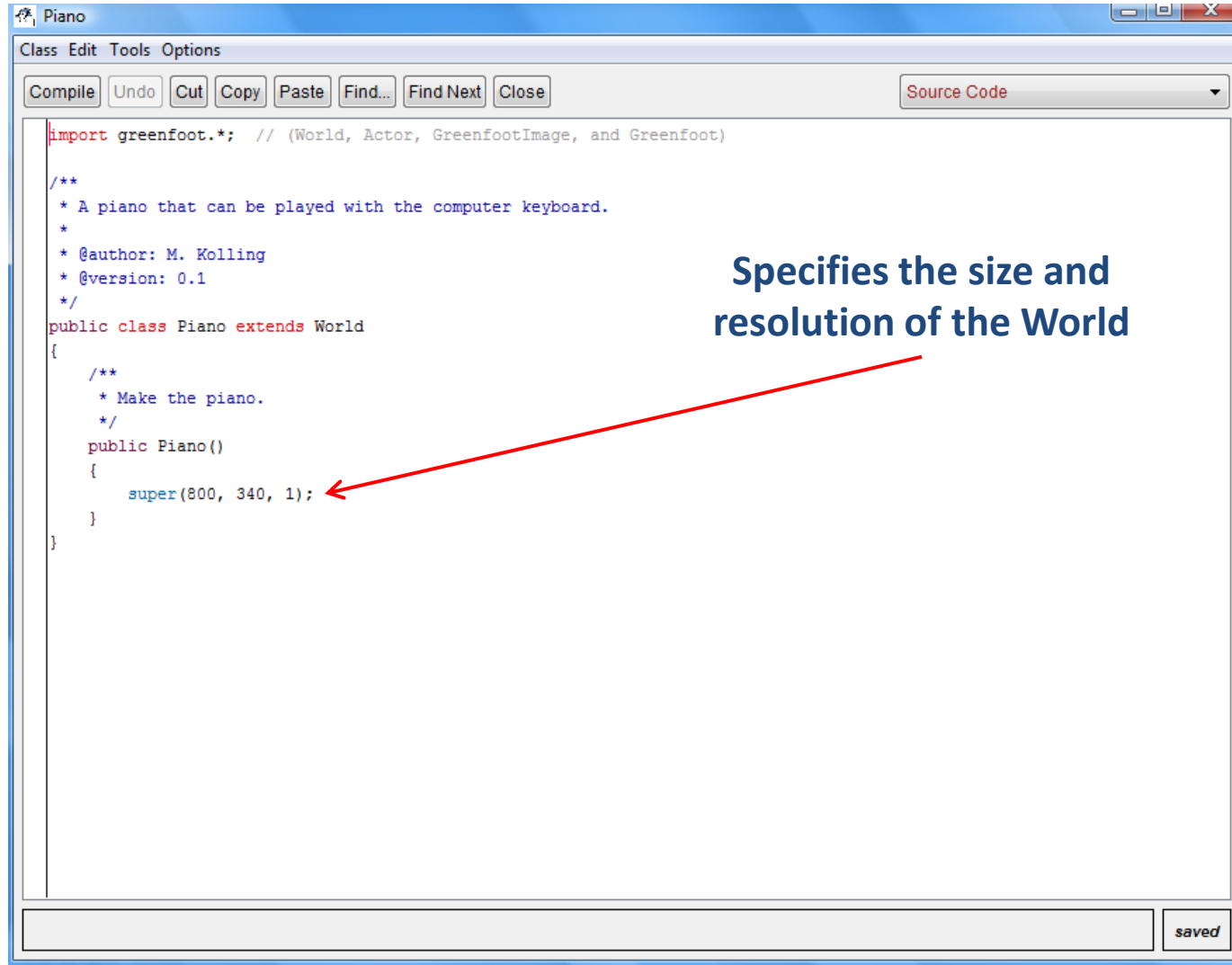
The Code is only Stubs

saved

Exercise 5.2



5.1 Animating the Key



The screenshot shows a Java IDE window titled "Piano". The window has a menu bar with "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Find Next", and "Close". There is also a "Source Code" dropdown menu. The main area contains the following Java code:

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A piano that can be played with the computer keyboard.
 *
 * @author: M. Kolling
 * @version: 0.1
 */
public class Piano extends World
{
    /**
     * Make the piano.
     */
    public Piano()
    {
        super(800, 340, 1);
    }
}
```

A red arrow points from the text "Specifies the size and resolution of the World" to the line `super(800, 340, 1);` in the constructor. At the bottom right of the window, there is a "saved" button.

Code 5.1

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

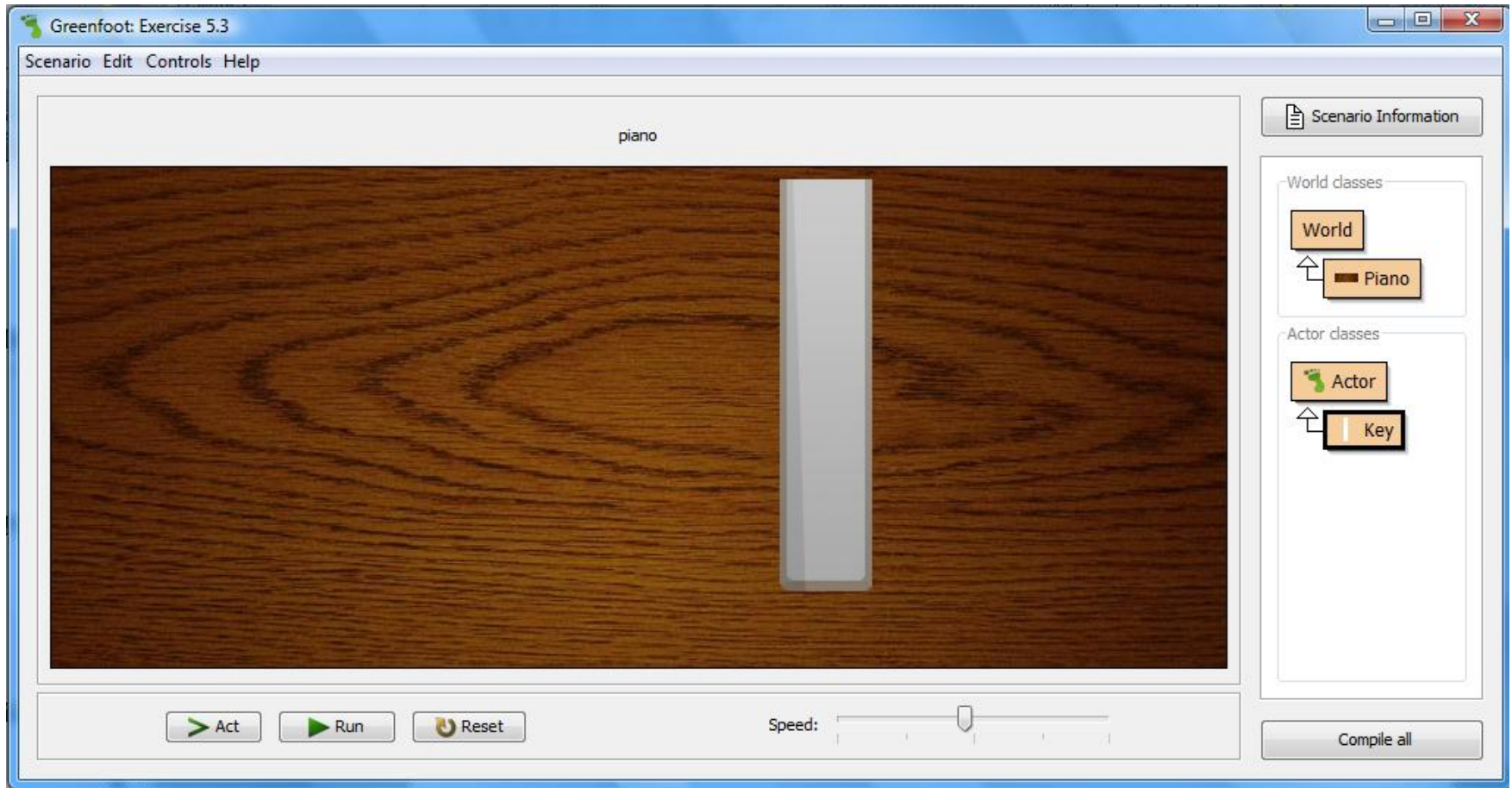
public class Key extends Actor
{
    /*
     * Create a new key.
     */
    public Key()
    {
    }

    /*
     * Do the action for this key.
     */
    public void act()
    {
    }
}
```

Code 5.2

```
public void act()  
{  
    if (Greenfoot.isKeyDown ("g"))  
    {  
        setImage ("white-key-down.png");  
    }  
    else  
    {  
        setImage ("white-key.png");  
    }  
}
```

Exercise 5.3



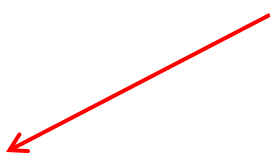
Code 5.3

```
public void act ()
{
    if ( !isDown && Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key-down.png");
        isDown = true;
    }

    if ( isDown && !Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key.png");
        isDown = false;
    }
}
```

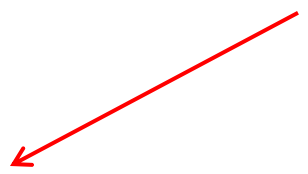
Logical Operations AND and NOT

if (not isDown and "g" is down)



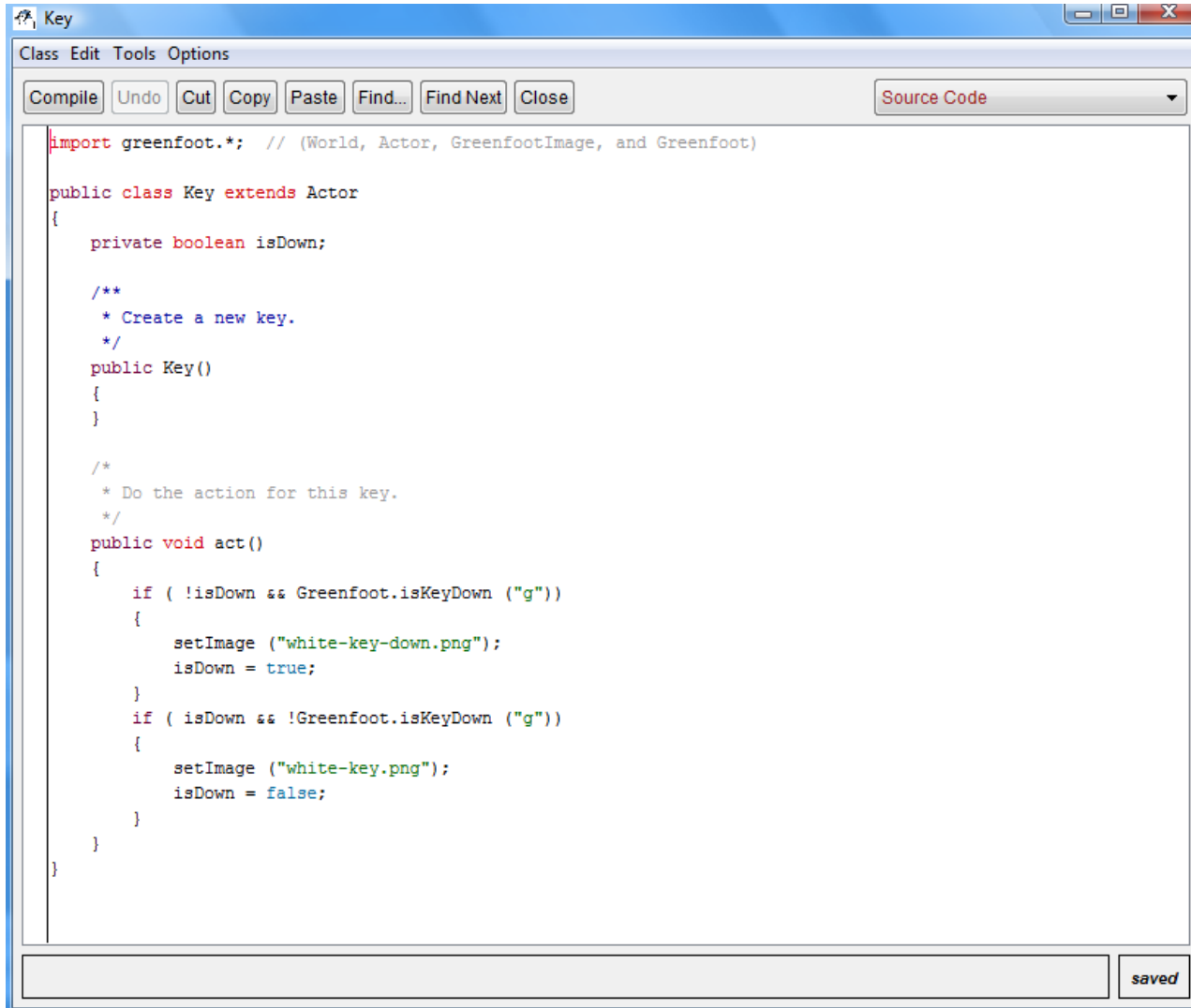
```
if ( !isDown && Greenfoot.isKeyDown ("g"))  
{  
    setImage ("white-key-down.png");  
    isDown = true;  
}
```

if (isDown and "g" is not down)



```
if ( isDown && !Greenfoot.isKeyDown ("g"))  
{  
    setImage ("white-key.png");  
    isDown = false;  
}
```

Exercise 5.4



```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

public class Key extends Actor
{
    private boolean isDown;

    /**
     * Create a new key.
     */
    public Key()
    {
    }

    /**
     * Do the action for this key.
     */
    public void act()
    {
        if ( !isDown && Greenfoot.isKeyDown ("g"))
        {
            setImage ("white-key-down.png");
            isDown = true;
        }
        if ( isDown && !Greenfoot.isKeyDown ("g"))
        {
            setImage ("white-key.png");
            isDown = false;
        }
    }
}
```

saved

5.2 Producing the Sound

The screenshot shows a Windows Explorer window with the address bar displaying the path: << ITP 100 (Greenfoot) HERE HERE HERE HERE >> Greenfoot > Greenfoot Exercises > Chapter 5 > Exercise 5.5 > sounds. The left sidebar shows the 'Folders' pane with the 'sounds' folder selected under 'Exercise 5.5'. The main pane displays a table of files with the following columns: Name, Artists, Album, #, Genre, and Rating. The files listed are WAV files named 2a.wav through 4g.wav, with some including sharps (e.g., 2a#.wav, 3a#.wav). Each file has a rating of five stars.

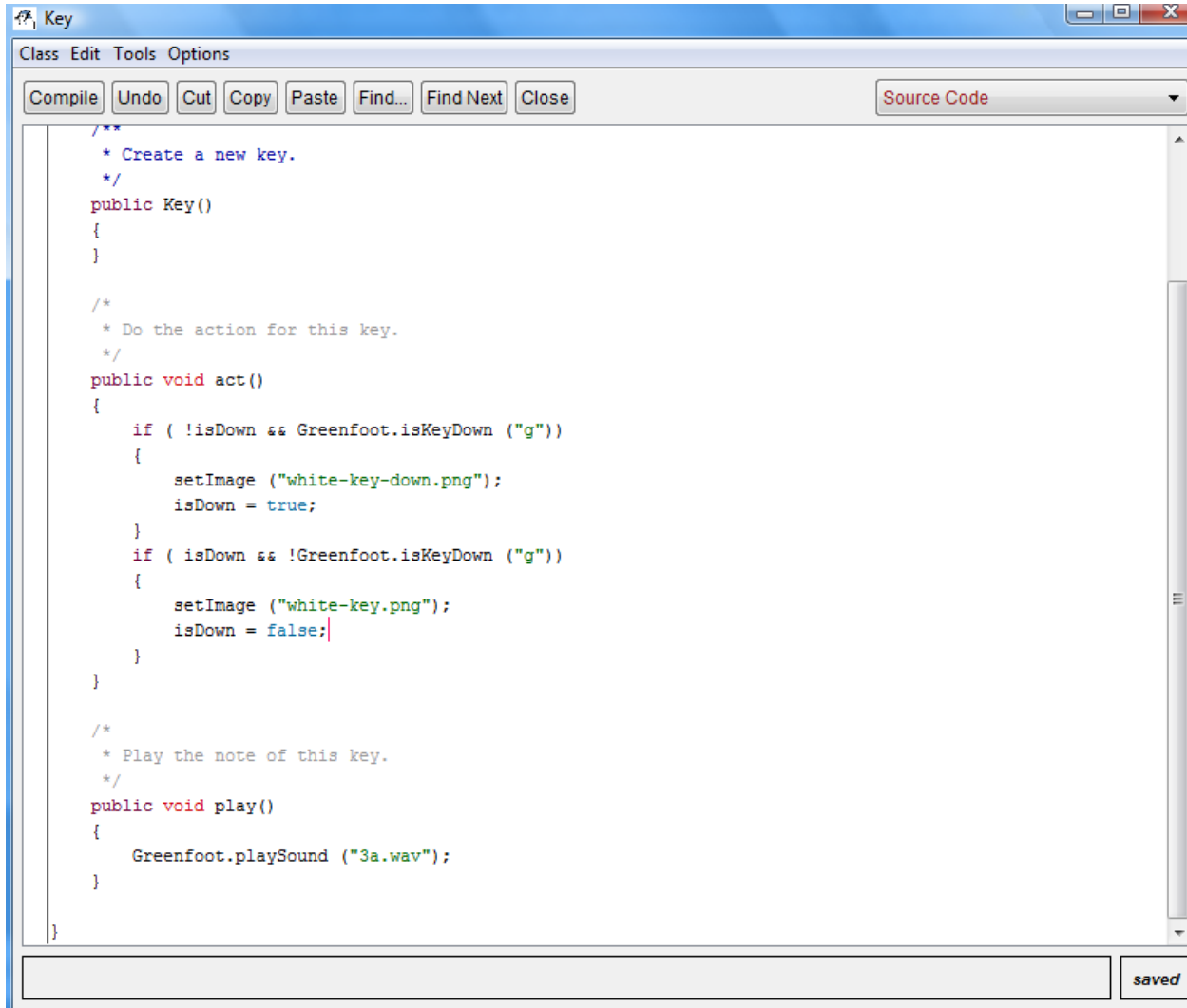
Name	Artists	Album	#	Genre	Rating
2a.wav					☆☆☆☆☆
2a#.wav					☆☆☆☆☆
2b.wav					☆☆☆☆☆
2c.wav					☆☆☆☆☆
2c#.wav					☆☆☆☆☆
2d.wav					☆☆☆☆☆
2d#.wav					☆☆☆☆☆
2e.wav					☆☆☆☆☆
2f.wav					☆☆☆☆☆
2f#.wav					☆☆☆☆☆
2g.wav					☆☆☆☆☆
2g#.wav					☆☆☆☆☆
3a.wav					☆☆☆☆☆
3a#.wav					☆☆☆☆☆
3b.wav					☆☆☆☆☆
3c.wav					☆☆☆☆☆
3c#.wav					☆☆☆☆☆
3d.wav					☆☆☆☆☆
3d#.wav					☆☆☆☆☆
3e.wav					☆☆☆☆☆
3f.wav					☆☆☆☆☆
3f#.wav					☆☆☆☆☆
3g.wav					☆☆☆☆☆
3g#.wav					☆☆☆☆☆
4c.wav					☆☆☆☆☆
4c#.wav					☆☆☆☆☆
4d.wav					☆☆☆☆☆
4d#.wav					☆☆☆☆☆
4e.wav					☆☆☆☆☆
4f.wav					☆☆☆☆☆
4f#.wav					☆☆☆☆☆
4g.wav					☆☆☆☆☆

The sounds folder has a collection of sound files each of which contains the sounds for a single piano key.

Code 5.4

```
/*  
 * Play the note of this key.  
 */  
public void play()  
{  
    Greenfoot.playSound ("3a.wav");  
}
```


Exercise 5.5



```
Key
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
/**
 * Create a new key.
 */
public Key()
{
}

/**
 * Do the action for this key.
 */
public void act()
{
    if ( !isDown && Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key-down.png");
        isDown = true;
    }
    if ( isDown && !Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key.png");
        isDown = false;
    }
}

/**
 * Play the note of this key.
 */
public void play()
{
    Greenfoot.playSound ("3a.wav");
}
}

saved
```

Exercise 5.6

Greenfoot: Exercise 5.5

Scenario Edit Controls Help

piano

Right Click on the Object
Click play

- inherited from Object* ▶
- inherited from Actor* ▶
- void act()
- void play()
- Inspect*
- Remove*

Scenario Information

World classes:

- World
- ↳ Piano

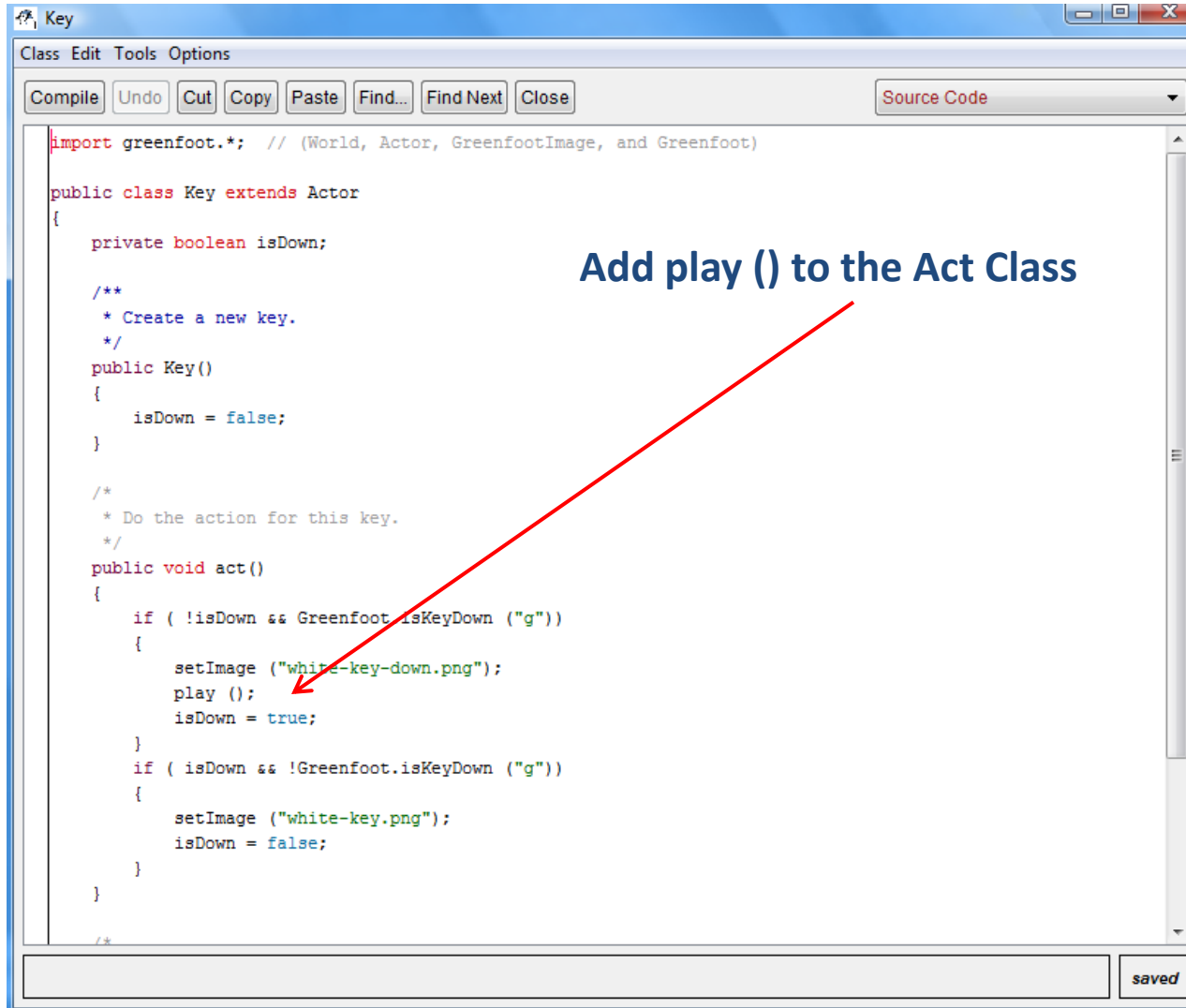
Actor classes:

- Actor
- ↳ Key

> Act ▶ Run ↻ Reset Speed: [Slider]

Compile all

Exercise 5.7



```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

public class Key extends Actor
{
    private boolean isDown;

    /**
     * Create a new key.
     */
    public Key()
    {
        isDown = false;
    }

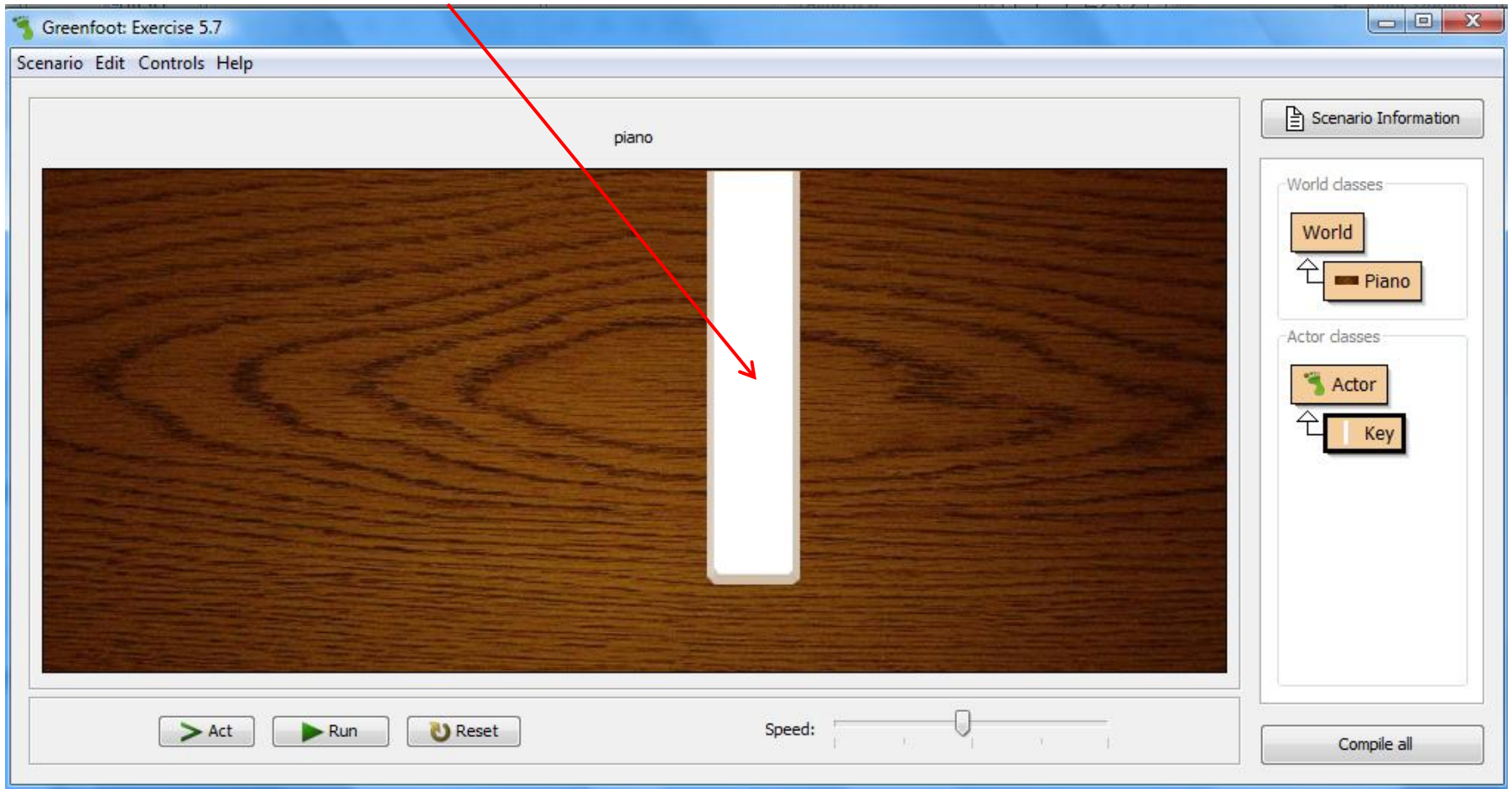
    /**
     * Do the action for this key.
     */
    public void act()
    {
        if ( !isDown && Greenfoot.isKeyDown ("g"))
        {
            setImage ("white-key-down.png");
            play ();
            isDown = true;
        }
        if ( isDown && !Greenfoot.isKeyDown ("g"))
        {
            setImage ("white-key.png");
            isDown = false;
        }
    }
}
```

Add play () to the Act Class

saved

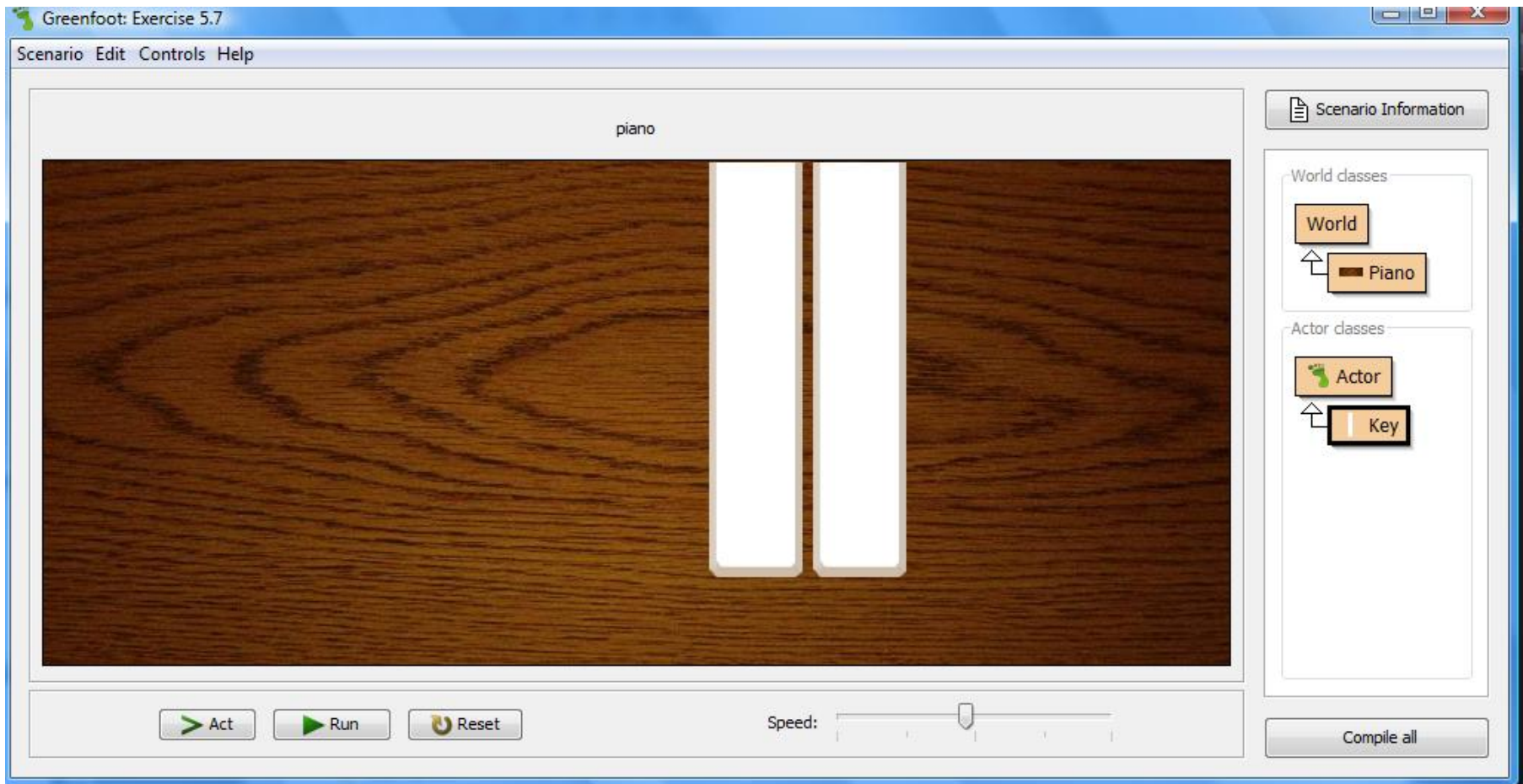
Exercise 5.7

Now the Key Plays a Note



Exercise 5.8

The Keys All React the Same Way



Code 5.5

```
public class Key extends Actor
{
    private boolean isDown;
    private String key;
    private String sound;

    /*
     * Create a new key linked to a given keyboard key, and
     * with a given sound.
     */
    public Key(String keyName, String soundFile)
    {
        key = keyName;
        sound = soundFile;
    }

    // methods omitted.
}
```

Exercise 5.9

```
public Key(String keyName, String soundFile)
{
    key = keyname;
    sound = soundFile;
}

/*
 * Do the action for this key.
 */
public void act()
{
    if ( !isDown && Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key-down.png");
        play ();
        isDown = true;
    }
    if ( isDown && !Greenfoot.isKeyDown ("g"))
    {
        setImage ("white-key.png");
        isDown = false;
    }
}

/*
 * Play the note of this key.
 */
public void play()
{
    Greenfoot.playSound ("3a.wav");
}
}
```

Change the "g" to key

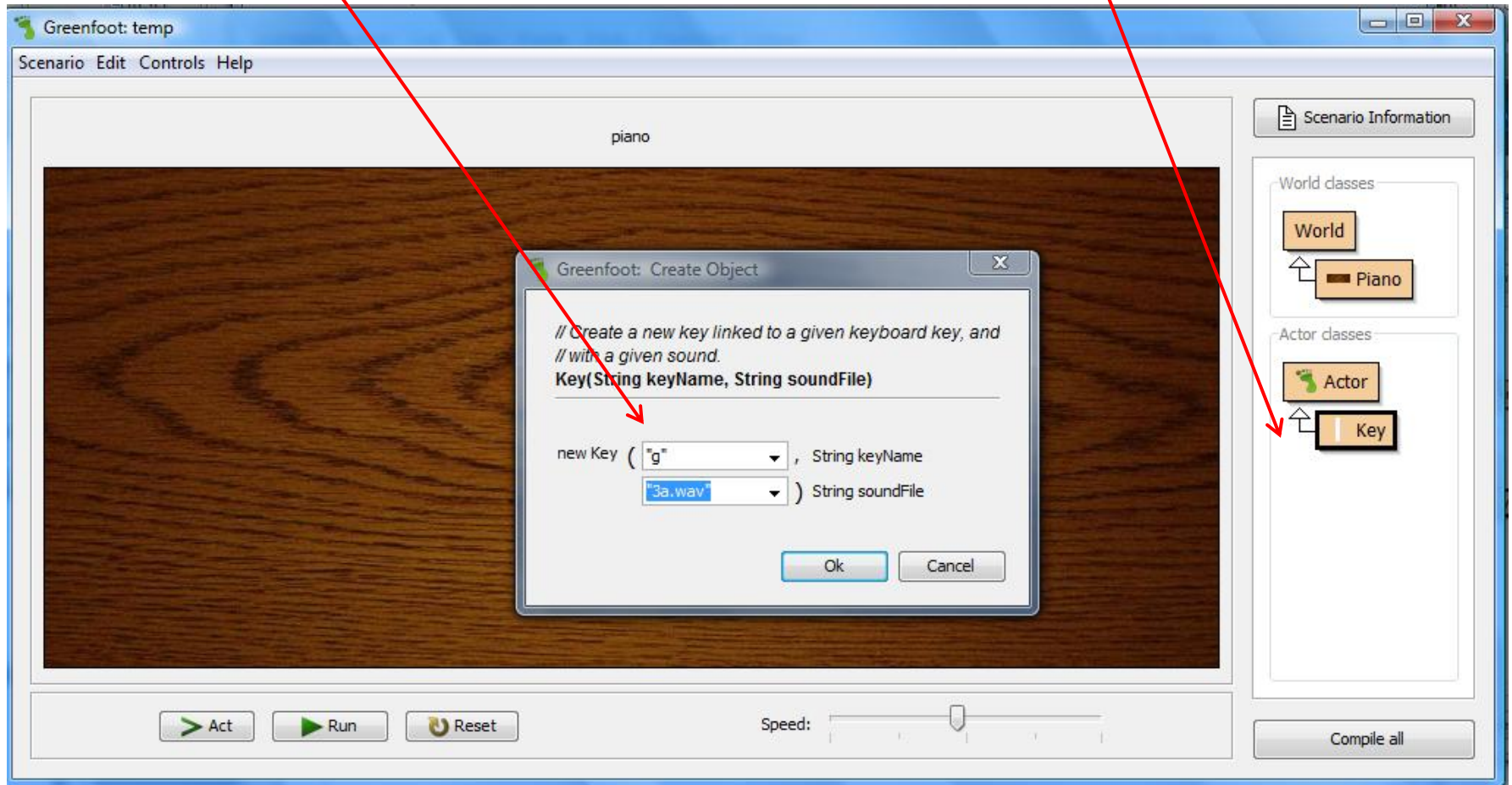
Change the "3a.wav" to sound

changed

Exercise 5.10

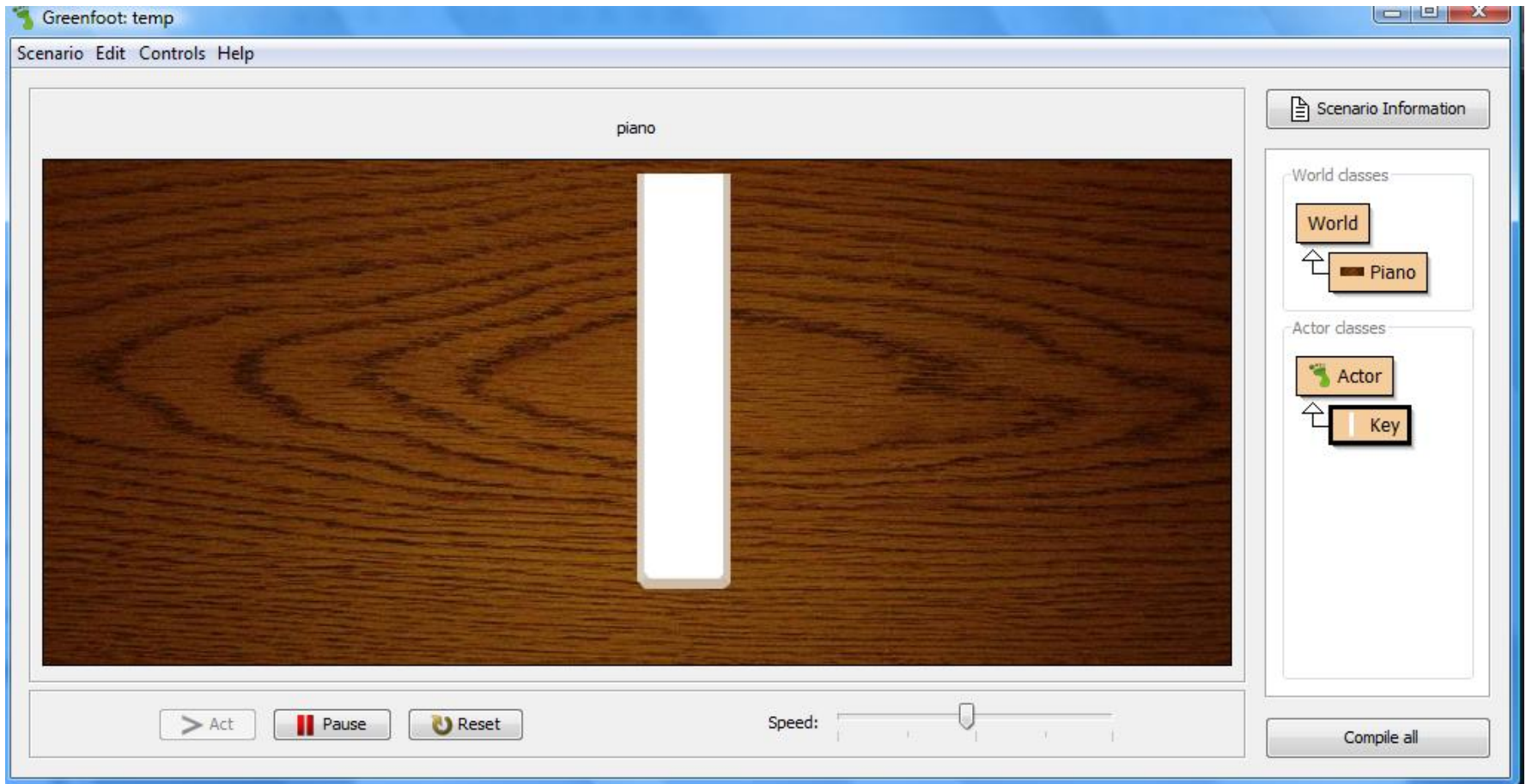
keyName is "g" and
soundFile is "3a.wav"

Right Click Key
Select new Key

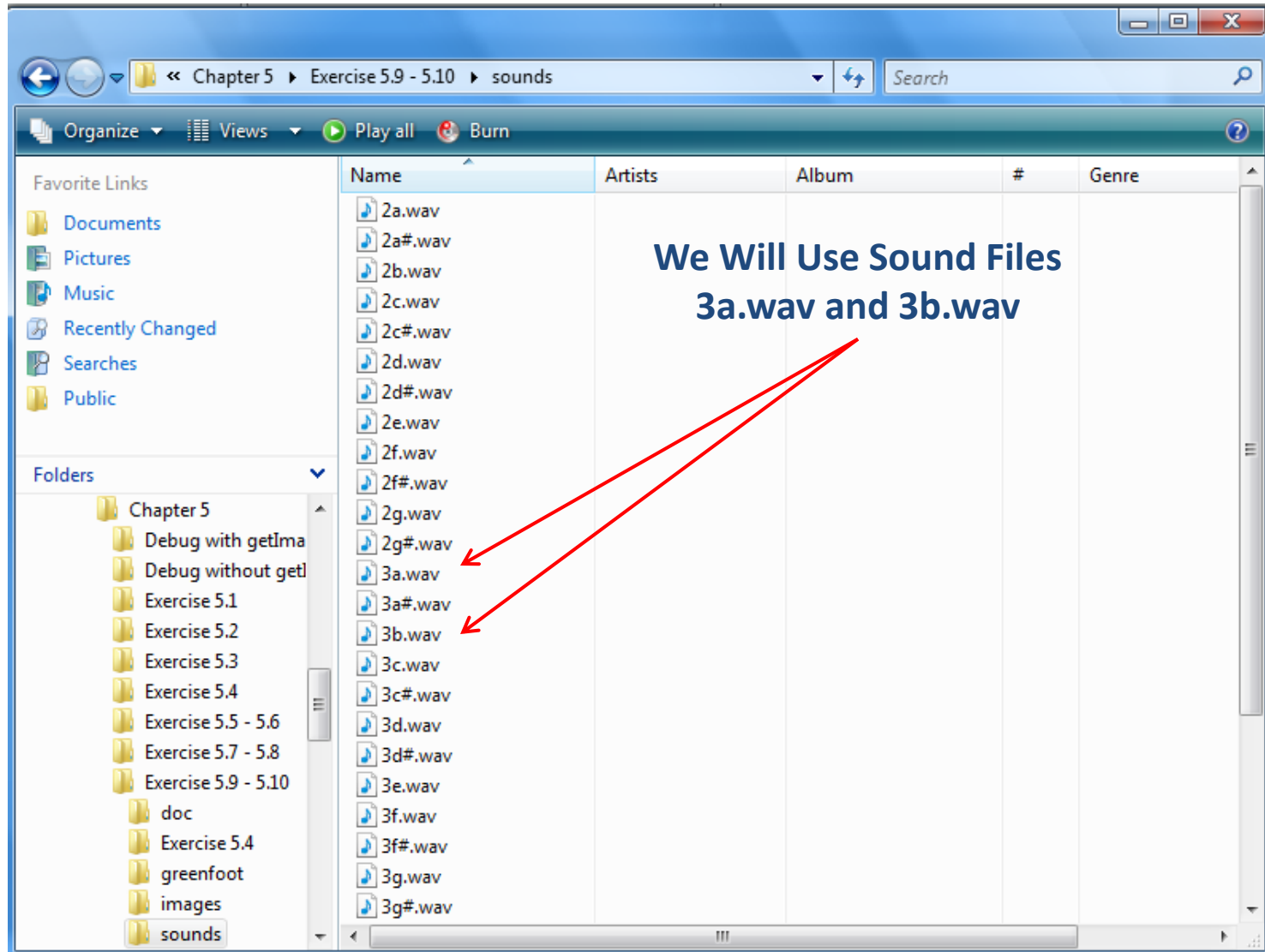


Exercise 5.10

Key Works as Before



sounds Folder



Exercise 5.10

Add the First Key

The screenshot shows the Greenfoot IDE interface. The main workspace displays a piano keyboard on a wooden surface. A dialog box titled "Greenfoot: Create Object" is open in the center. The dialog contains the following text:

```
// Create a new key linked to a given keyboard key, and  
// with a given sound.  
Key(String keyName, String soundFile)
```

Below the code, there are two dropdown menus for creating a new object:

```
new Key ( "g" , String keyName  
         "3a.wav" ) String soundFile
```

The dialog has "Ok" and "Cancel" buttons at the bottom.

On the right side of the IDE, there is a "Scenario Information" panel. It shows "World classes" with "World" and "Piano" (inherited from "World"). It also shows "Actor classes" with "Actor" and "Key" (inherited from "Actor").

At the bottom of the IDE, there are buttons for "Act", "Run", and "Reset", along with a "Speed" slider and a "Compile all" button.

Exercise 5.10

Greenfoot: Exercise 5.9 - 5.10

Scenario Edit Controls Help

piano

Add the Second Key

Greenfoot: Create Object

```
// Create a new key linked to a given keyboard key, and  
// with a given sound.  
Key(String keyName, String soundFile)
```

new Key ("h" , String keyName
"3b.wav") String soundFile

Ok Cancel

Scenario Information

World classes

- World
- Piano

Actor classes

- Actor
- Key

Act Run Reset

Speed: [Slider]

Compile all

Exercise 5.10

The screenshot displays the Greenfoot IDE interface for 'Exercise 5.9 - 5.10'. The main workspace shows a piano keyboard with two white keys highlighted. The text 'We now Have a Piano with Two Keys' is displayed in green on the piano's surface. The top menu bar includes 'Scenario', 'Edit', 'Controls', and 'Help'. The right sidebar contains a 'Scenario Information' panel with 'World classes' (World and Piano) and 'Actor classes' (Actor and Key). The bottom control bar features 'Act', 'Run', and 'Reset' buttons, a 'Speed' slider, and a 'Compile all' button.

Greenfoot: Exercise 5.9 - 5.10

Scenario Edit Controls Help

piano

We now Have a Piano with Two Keys

Scenario Information

World classes

- World
- Piano

Actor classes

- Actor
- Key

Act Run Reset

Speed: [Slider]

Compile all

5.4 Building the Piano

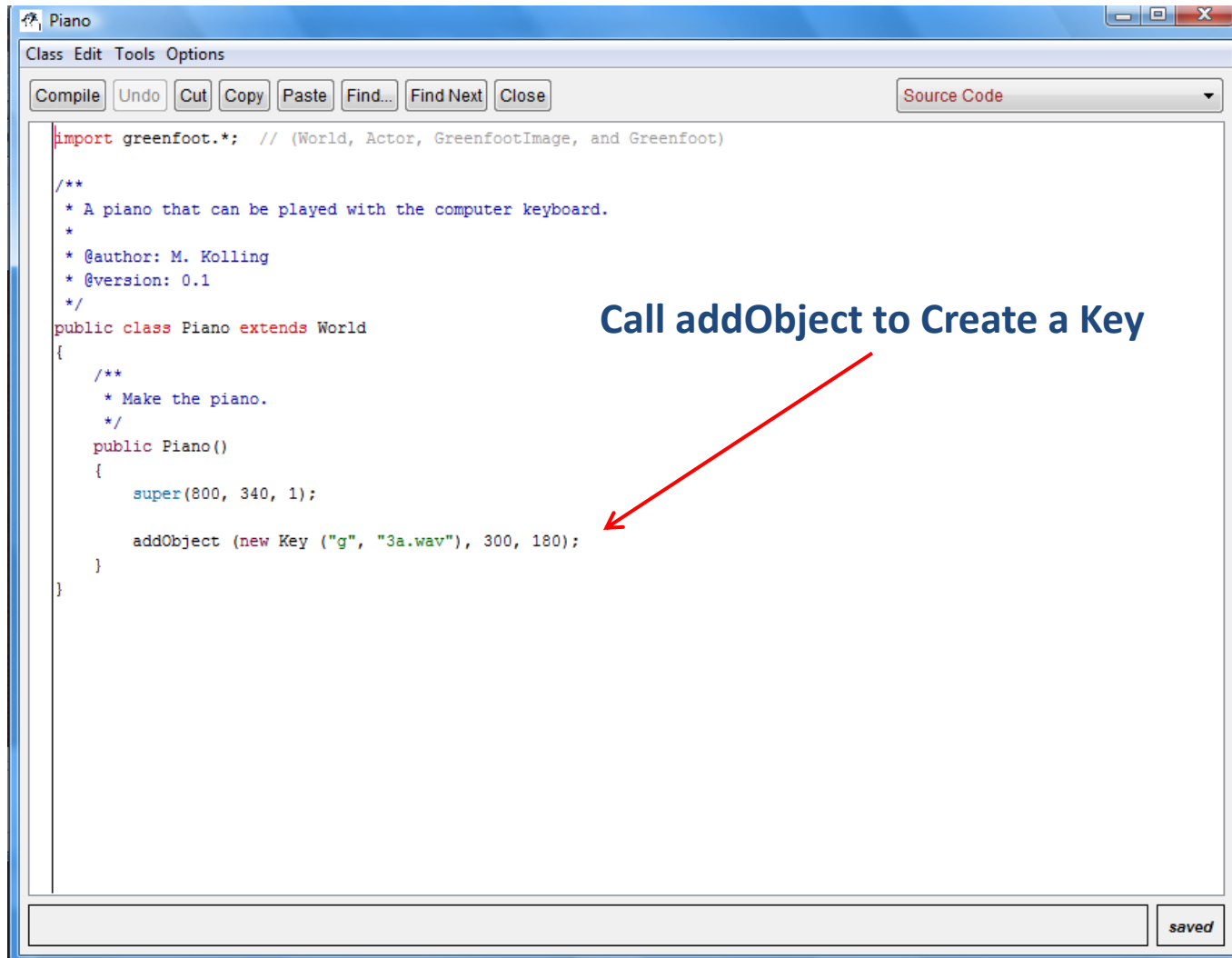
```
addObject ( new Key ( "g", "3a.wav", 300, 180 );
```

Remember that the expression

```
new Key ( "g", "3a.wav" )
```

creates a new Key object with a specific key and a sound file.

Exercise 5.11



```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A piano that can be played with the computer keyboard.
 *
 * @author: M. Kolling
 * @version: 0.1
 */
public class Piano extends World
{
    /**
     * Make the piano.
     */
    public Piano()
    {
        super(800, 340, 1);

        addObject (new Key ("g", "3a.wav"), 300, 180);
    }
}
```

Call addObject to Create a Key

saved

Exercise 5.11

The screenshot shows the Greenfoot IDE interface for Exercise 5.11. The main window displays a piano key on a wooden background. The right sidebar shows the class hierarchy for the scenario, including World classes (World, Piano) and Actor classes (Actor, Key). The bottom toolbar contains buttons for Act, Run, Reset, a Speed slider, and a Compile all button.

Greenfoot: Exercise 5.11

Scenario Edit Controls Help

piano

Scenario Information

World classes

- World
- Piano

Actor classes

- Actor
- Key

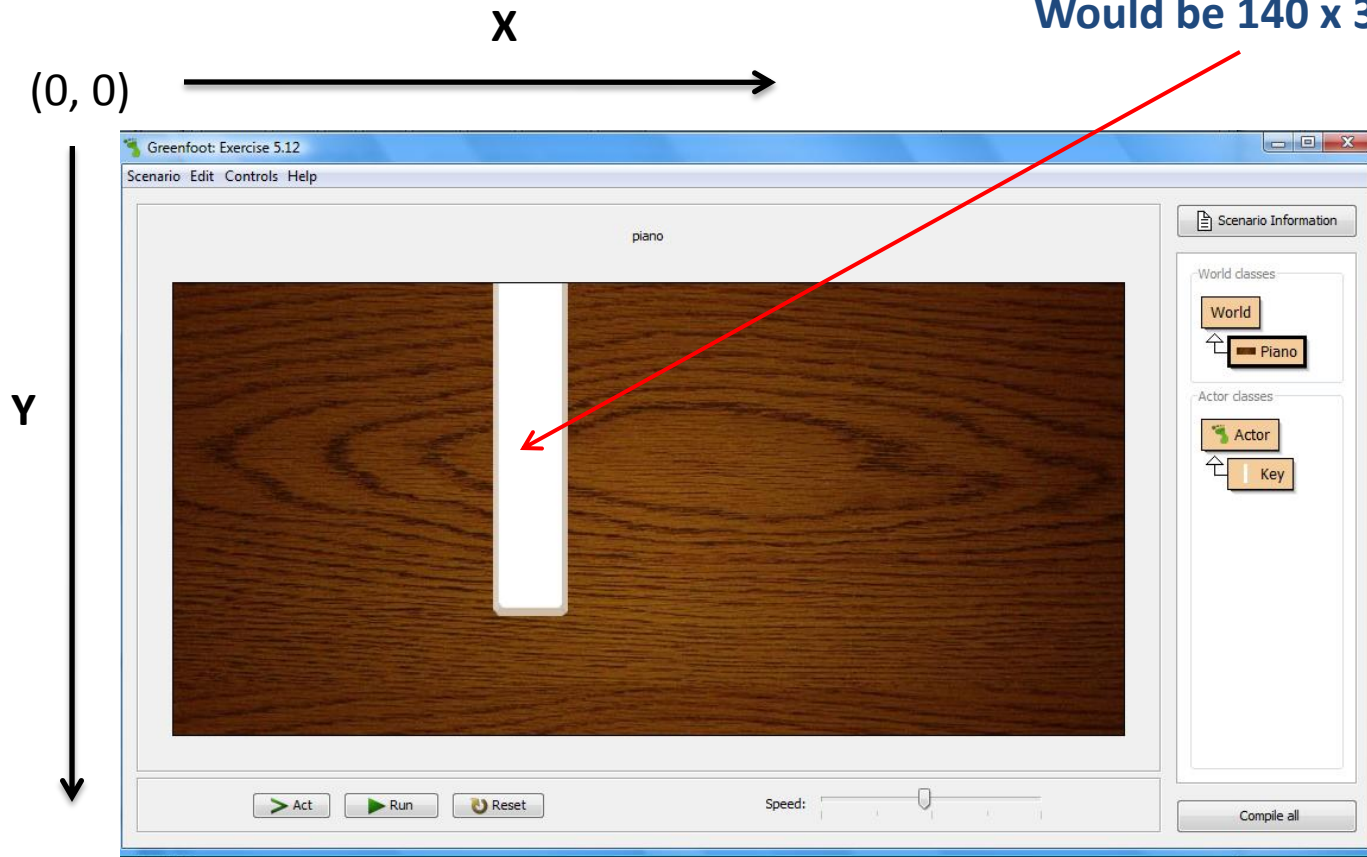
Act Run Reset

Speed: [Slider]

Compile all

Exercise 5.12

The Key is 280 x 63
Therefore the Center of the Key
Would be 140 x 31 ½



Exercise 5.12

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)
```

```
/*
 * A piano that can be played with the computer keyboard.
 *
 * @author: M. Kolling
 * @version: 0.1
 */
public class Piano extends World
{
    /*
     * Make the piano.
     */
    public Piano()
    {
        super(800, 380, 1);

        addObject (new Key ("g", "3a.wav"), 300, 140);
    }
}
```

Exercise 5.13

The screenshot shows the Greenfoot IDE interface for 'Exercise 5.13'. The main workspace displays a piano keyboard on a wooden background. Two keys are highlighted in white. To the right of the keys, the following text is displayed in green:

$31 \frac{1}{2} + 31 \frac{1}{2} = 63$
The First Key is Located at 300
Therefore $300 - 63 = 237$
Locate the Second Key at 237

The interface includes a menu bar (Scenario, Edit, Controls, Help), a 'Scenario Information' panel on the right showing class hierarchies for 'World' and 'Piano' (World classes) and 'Actor' and 'Key' (Actor classes), and a control bar at the bottom with 'Act', 'Pause', 'Reset' buttons, a 'Speed' slider, and a 'Compile all' button.

Exercise 5.13

```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)
```

```
/*  
 * A piano that can be played with the computer keyboard.  
 *  
 * @author: M. Kolling  
 * @version: 0.1  
 */  
public class Piano extends World  
{  
    /**  
     * Make the piano.  
     */  
    public Piano()  
    {  
        super(800, 380, 1);  
  
        addObject (new Key ("g", "3a.wav"), 300, 140);  
        addObject (new Key ("f", "3g.wav"), 237, 140);  
    }  
}
```

Exercise 5.14

```
public class Piano extends World
{
    /*
     * Make the piano.
     */
    public Piano()
    {
        super(800, 380, 1);
        makeKeys ();
    }

    /*
     * Create the Piano Keys.
     */
    public void makeKeys()
    {
        addObject (new Key ("g", "3a.wav"), 300, 140);
        addObject (new Key ("f", "3g.wav"), 237, 140);
    }
}
```

Loops

- Repetition in programs allows us to repeat something over and over.
- We achieve repetition through loops.
- A loop is a statement in programming languages that can execute a section of code multiple times.
- We will look at a while loop to help us repeat.

While-loop

- This will keep looping until the condition indicated on the loop is false.

```
while (/*booleanExpression*/)
{
    //code that should be repeated
}
```

While-Loop

- In order to help us keep track of how many times we are looping, we need to create a variable to store a count.
- Inside the loop, we also must remember to increment the count so that the loop executes the correct number of times.

While loop

```
while (true)
{
    //code that should be repeated
}
```

- This loop will continue forever because true is always true.

5.5 Loops: The While Loop

```
while ( condition )  
{  
    statement;  
    statement;  
    . . .  
}
```

While Loop

```
int count = 0;
while (count < 10)
{
    //code that should be repeated
    count = count + 1;
}
```

The code in this loop will execute 10 times

Local Variables

```
int i = 0;

while ( i < 100 )
{
    statement;
    statement;
    . . .
    i = i + 1;
}
```

Local Variable

- A local variable is declared inside a method body, not at the beginning of the class
- It has no visibility modifier (private or public) in front of it
- It exists only until the current method finishes running, then it will be erased

for Loop Better Than while

```
int i = 0;
while ( i < 100 )
{
    statement;
    statement;
    . . .
    i = i + 1;
}
```

```
int i;
for ( i=0; i < 100; i++)
{
    statement;
    statement;
    . . .
}
```

Exercise 5.15

```
/*  
 * Create the Piano Keys  
 */  
public void makeKeys()  
{  
    int i;  
  
    for (i=0; i<12; i++)  
        addObject (new Key ("g", "3a.wav"), 300, 140);  
}
```

Exercise 5.15

The screenshot shows the Greenfoot IDE interface for 'Exercise 5.15'. The main workspace displays a piano keyboard on a wooden background. The text 'It Appears That There is Only One But The Keys Are Stacked Upon One Another' is written in green on the right side of the piano. The word 'piano' is centered above the keyboard. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar with 'Act', 'Run', and 'Reset' buttons, a 'Speed' slider, and a 'Scenario Information' panel on the right. The 'Scenario Information' panel shows 'World classes' (World and Piano) and 'Actor classes' (Actor and Key).

Greenfoot: Exercise 5.15

Scenario Edit Controls Help

piano

It Appears That There is Only One
But The Keys Are Stacked Upon
One Another

World classes

- World
- Piano

Actor classes

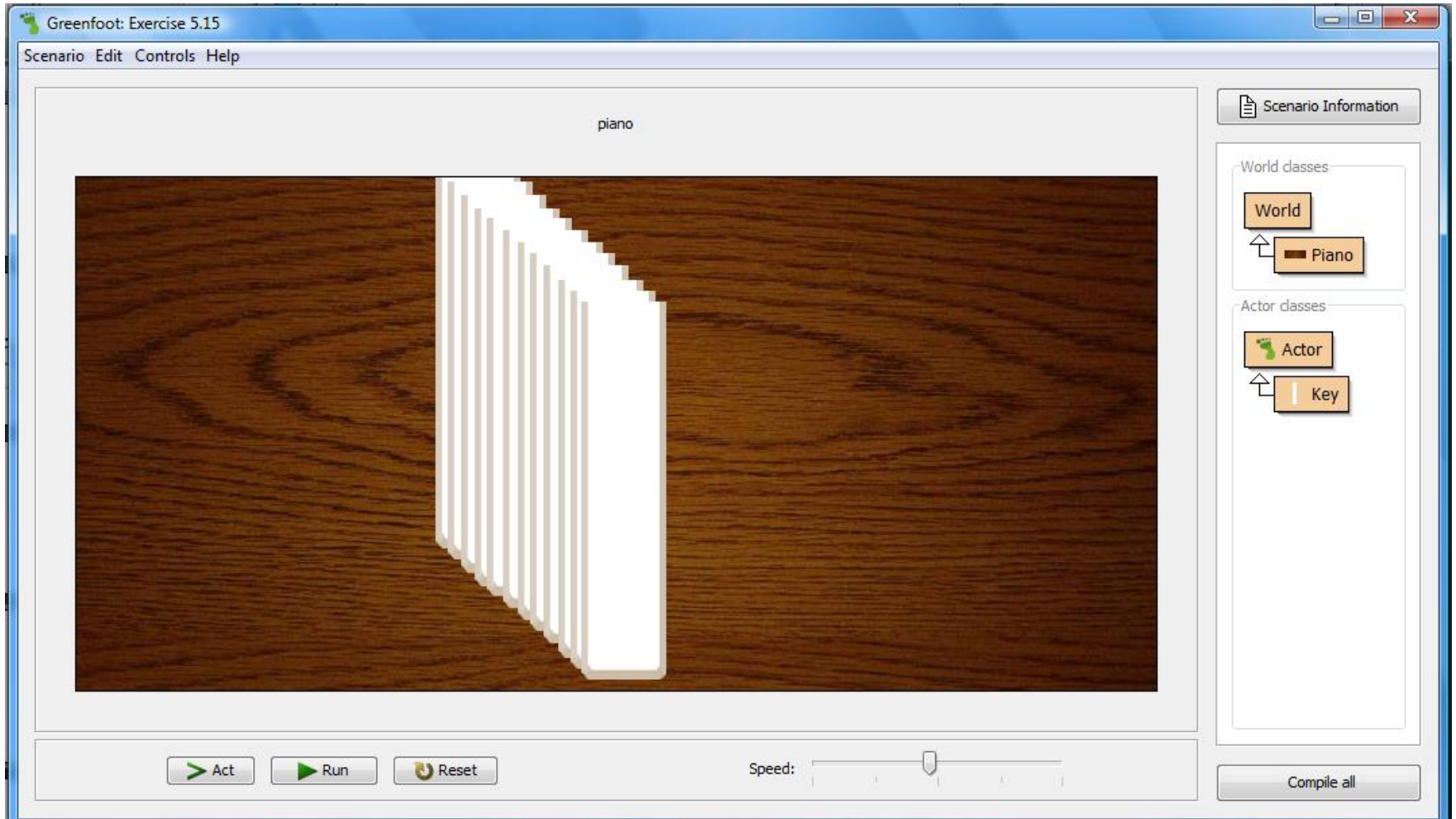
- Actor
- Key

Act Run Reset

Speed: [Slider]

Compile all

Exercise 5.15



Exercise 5.16

```
/*  
 * Create the Piano Keys  
 */  
Public void makeKeys()  
{  
    int i;  
  
    for (i=0; i<12; i++)  
        addObject (new Key ("g", "3a.wav"), i*63, 140);  
}
```

Exercise 5.16

The screenshot shows the Greenfoot IDE interface for 'Exercise 5.16'. The main workspace displays a piano keyboard with the following text overlaid:

Key Width is $63 * 12 \text{ Keys} = 756$
World Width is $800 - 756 = 44$
Half the Space on Either Side $44 / 2 = 22$
Space at Edge $22 + \text{Half a Key Width} 31 \frac{1}{2} = 53 \frac{1}{2}$

The right-hand panel shows the class hierarchy:

- World classes:
 - World
 - Piano (inherits from World)
- Actor classes:
 - Actor
 - Key (inherits from Actor)

The bottom of the window features control buttons: 'Act', 'Run', 'Reset', a 'Speed' slider, and a 'Compile all' button.

Exercise 5.17

```
for (i=0; i<12; i++)  
    addObject (new Key ("g", "3a.wav"), i*63 + 54, 140);
```

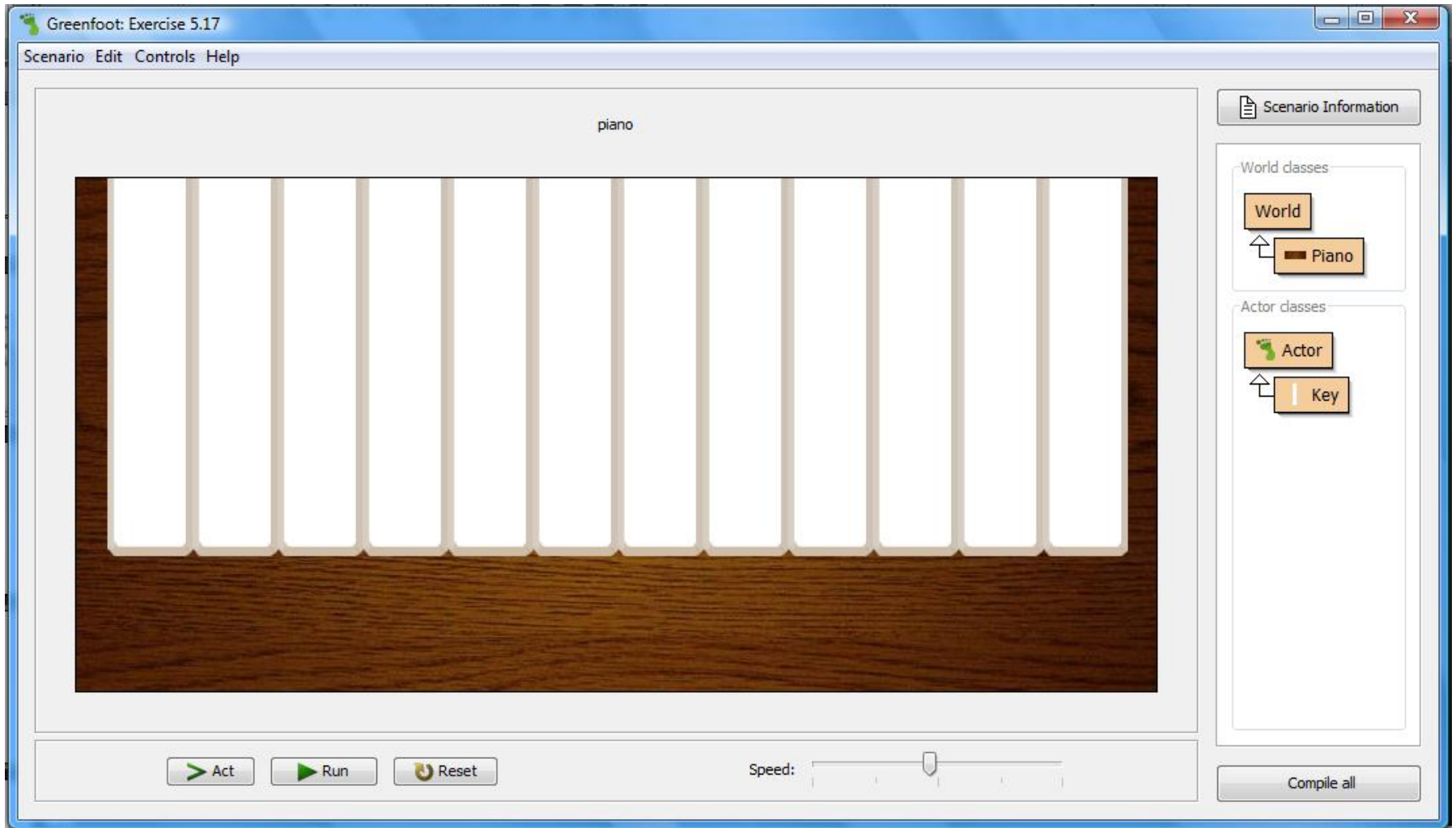
The for Loop Will Execute 12 Times

The Values for i Will Be 0, 1, . . . , 11

Exercise 5.17

```
/*  
 * Create the Piano Keys  
 */  
public void makeKeys()  
{  
    int i;  
  
    for (i=0; i<12; i++)  
        addObject (new Key ("g", "3a.wav"), i*63 + 54, 140);  
}
```

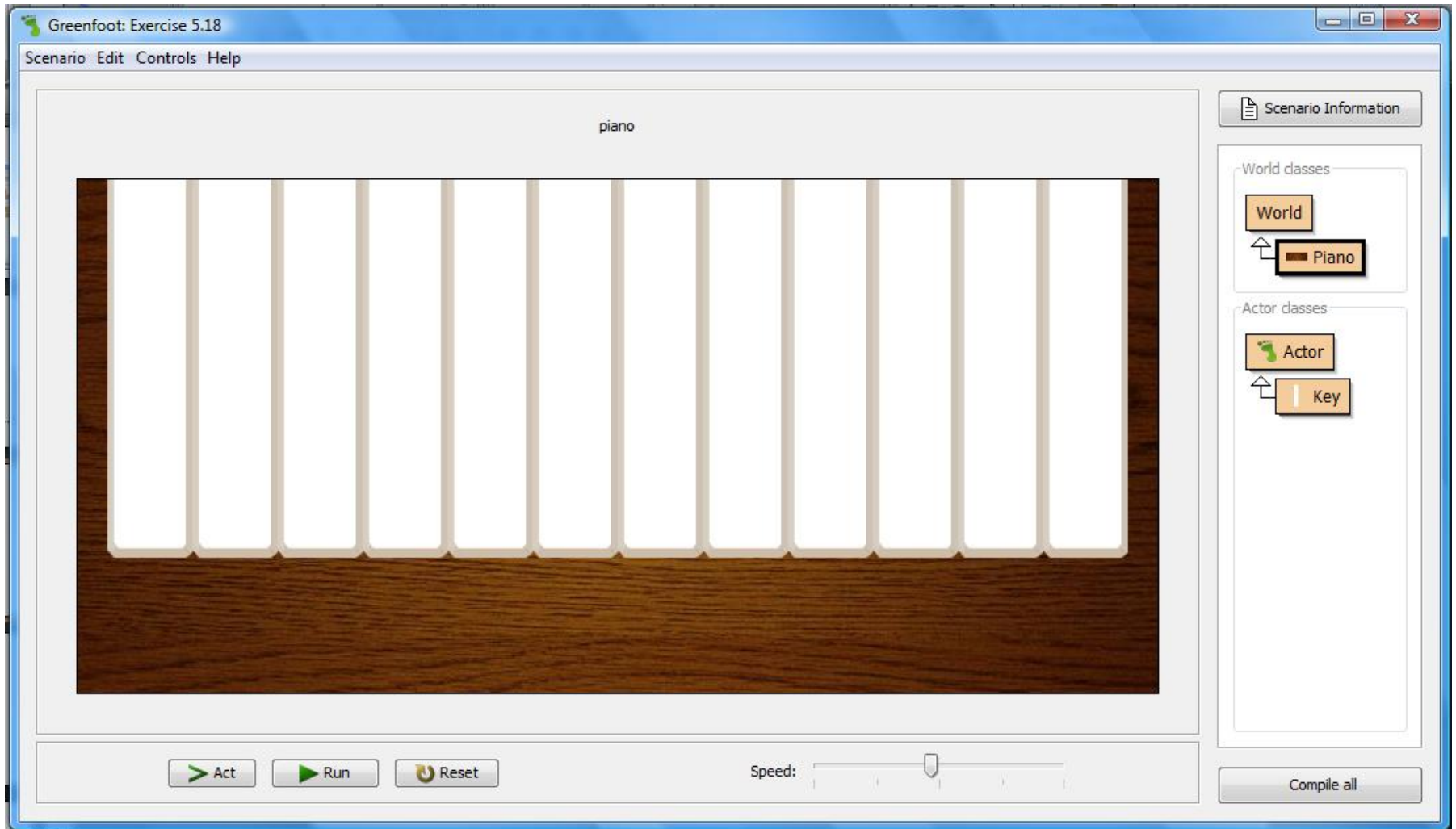
Exercise 5.17



Exercise 5.18

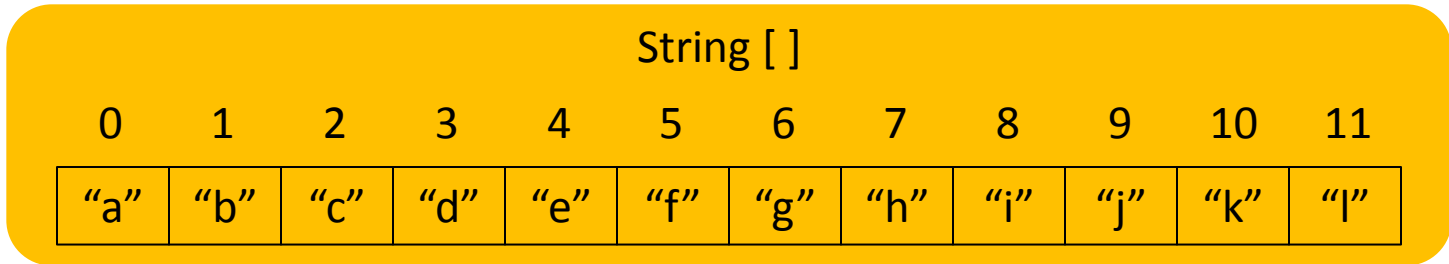
```
/*  
 * Create the Piano Keys  
 */  
public void makeKeys()  
{  
    int i;  
    int keyWidth;  
    int keyHeight;  
    int spaceAtEdge;  
  
    Key key = new Key(" ", " ");  
    keyWidth = key.getImage().getWidth();  
    keyHeight = key.getImage().getHeight();  
    spaceAtEdge = (800 - keyWidth*12) / 2;  
  
    for (i=0; i<12; i++)  
        addObject (new Key ("g", "3a.wav"), keyWidth*i + spaceAtEdge + keyWidth/2, keyHeight / 2);  
}
```

Exercise 5.18



5.6 Using Arrays

String [] names



```
String [] names;
```

```
names = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l"};
```

names [3] contains the string "d"

Code 5.6

```
public class Piano extends World
{
    private String[] whiteKeys =
        { "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "\\\" };
    private String[] whiteNotes =
        { "3c", "3d", "3e", "3f", "3g", "3a", "3b", "4c", "4d", "4e", "4f", "4g" };

    // constructor and methods omitted
}
```

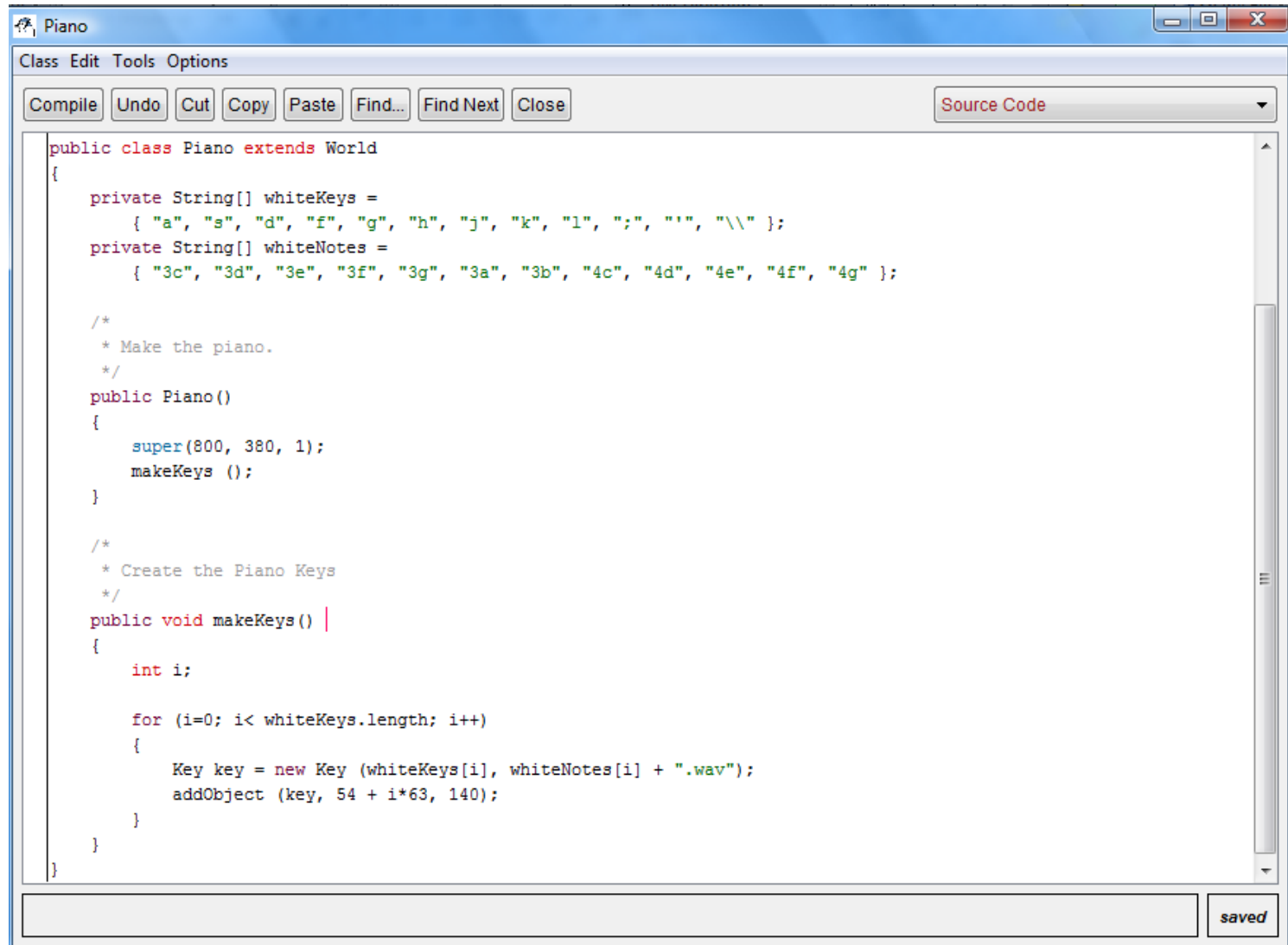
Code 5.7

```
/*  
 * Create the Piano Keys  
 */  
public void makeKeys()  
{  
    int i;  
  
    for (i=0; i < whiteKeys.length; i++)  
    {  
        Key key = new Key (whiteKeys[i], whiteNotes[i] + ".wav");  
        addObject (key, 54 + i*63, 140);  
    }  
}
```

We moved the creation of a new key out of the addObject method

Use a plus symbol (+) to concatenate whiteNotes[i] with ".wav"

Exercise 5.19



```
public class Piano extends World
{
    private String[] whiteKeys =
        { "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "\\\" };
    private String[] whiteNotes =
        { "3c", "3d", "3e", "3f", "3g", "3a", "3b", "4c", "4d", "4e", "4f", "4g" };

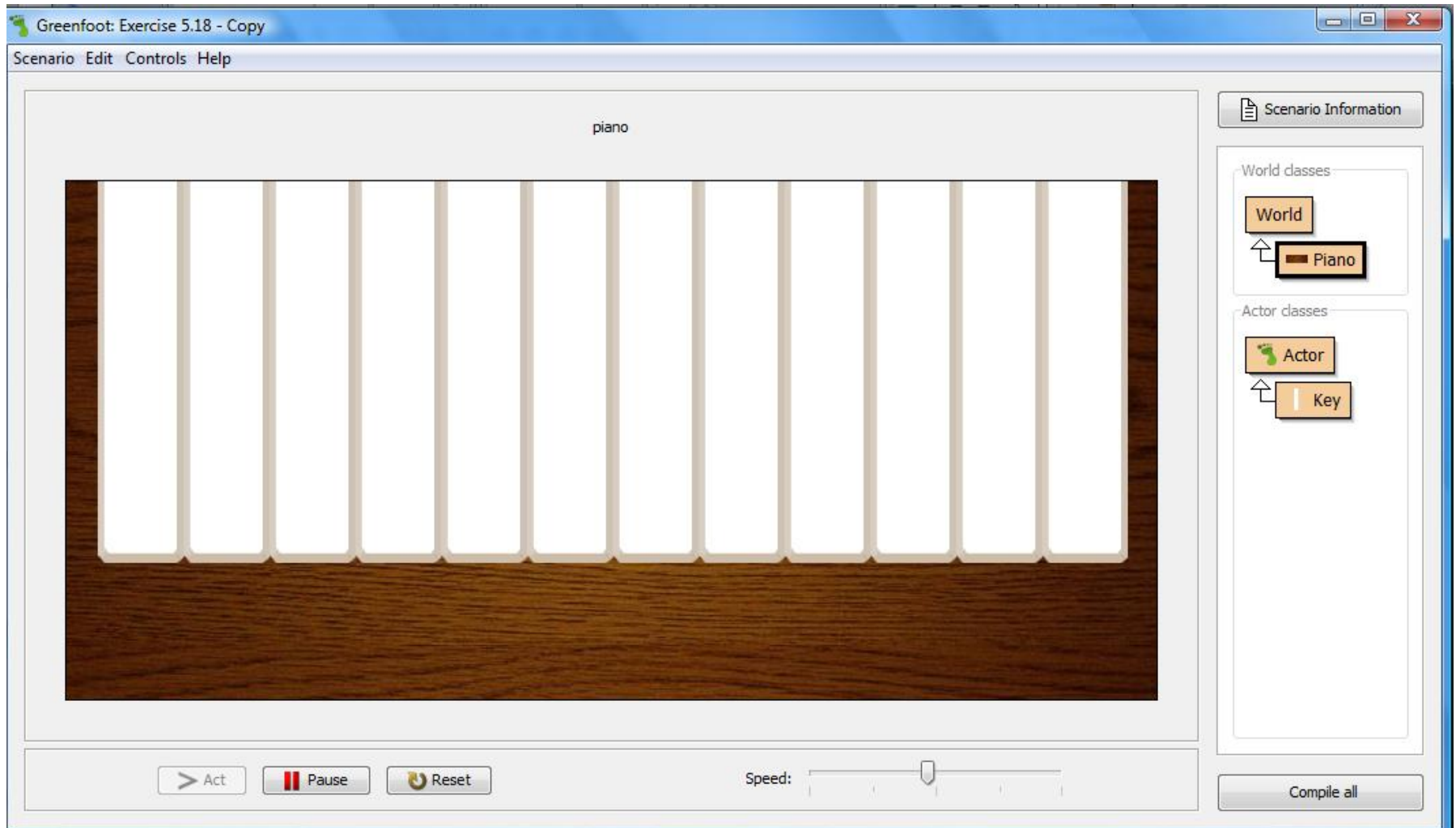
    /*
     * Make the piano.
     */
    public Piano()
    {
        super(800, 380, 1);
        makeKeys ();
    }

    /*
     * Create the Piano Keys
     */
    public void makeKeys() |
    {
        int i;

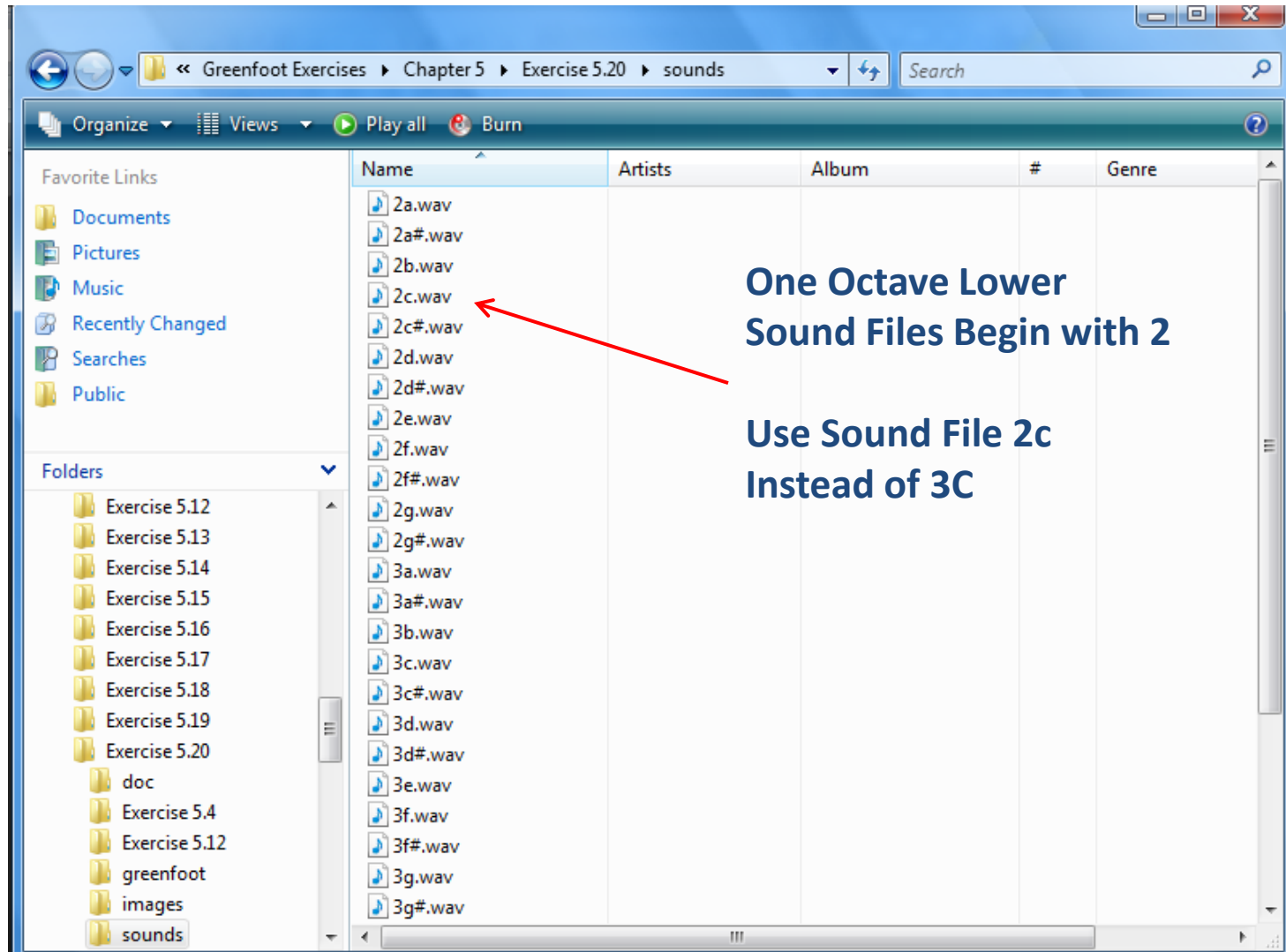
        for (i=0; i< whiteKeys.length; i++)
        {
            Key key = new Key (whiteKeys[i], whiteNotes[i] + ".wav");
            addObject (key, 54 + i*63, 140);
        }
    }
}
```

saved

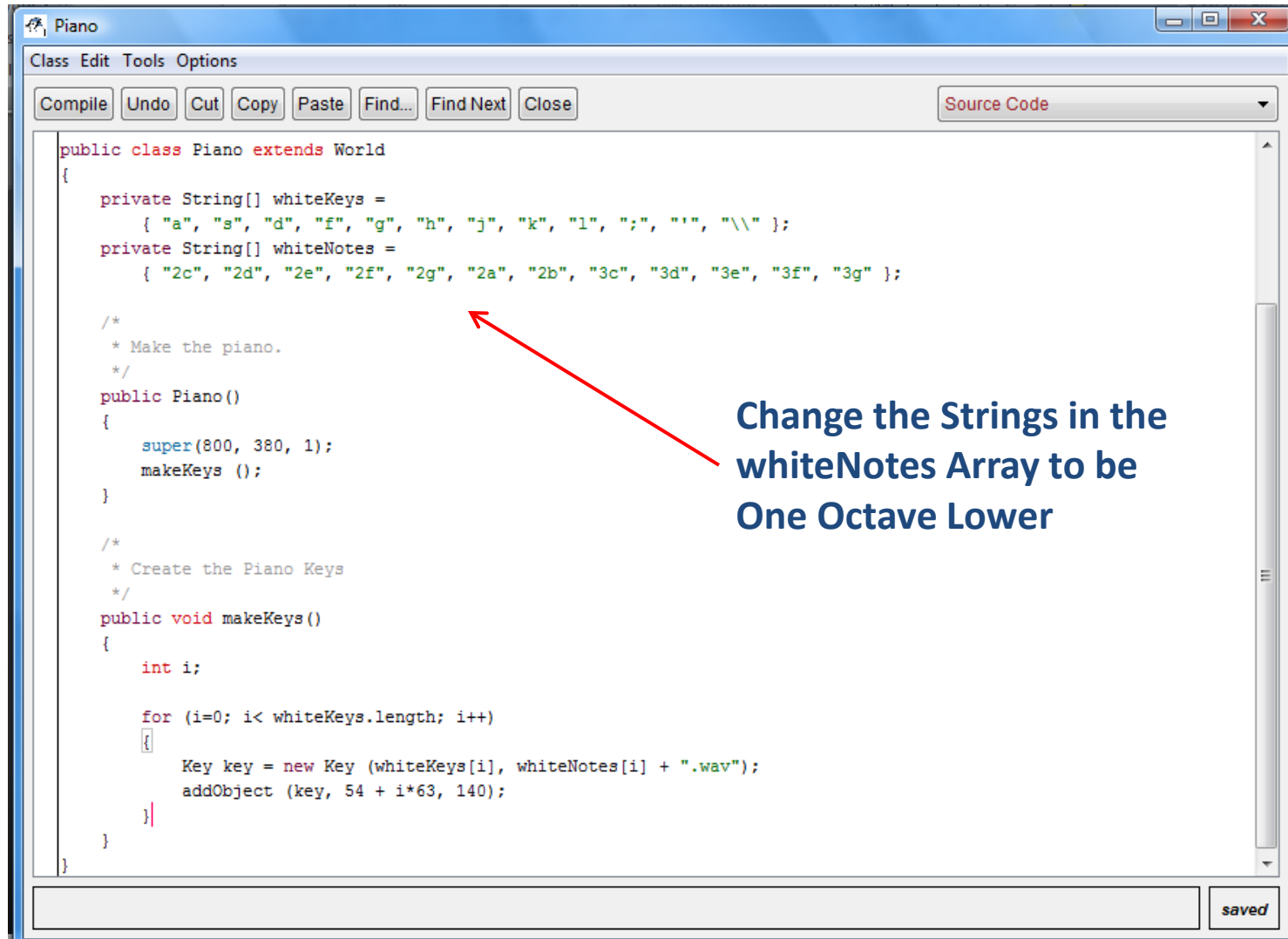
Exercise 5.19



Exercise 5.20



Exercise 5.20



```
public class Piano extends World
{
    private String[] whiteKeys =
        { "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "\\\" };
    private String[] whiteNotes =
        { "2c", "2d", "2e", "2f", "2g", "2a", "2b", "3c", "3d", "3e", "3f", "3g" };

    /*
     * Make the piano.
     */
    public Piano()
    {
        super(800, 380, 1);
        makeKeys ();
    }

    /*
     * Create the Piano Keys
     */
    public void makeKeys()
    {
        int i;

        for (i=0; i< whiteKeys.length; i++)
        {
            Key key = new Key (whiteKeys[i], whiteNotes[i] + ".wav");
            addObject (key, 54 + i*63, 140);
        }
    }
}
```

Change the Strings in the whiteNotes Array to be One Octave Lower

saved

Exercise 5.22

```
public Key (String keyName, String soundFile)
{
    key = keyName;
    sound = soundFile;
}
```

Exercise 5.22

```
public Key (String keyName, String soundFile, String img1, String img2)
{
    key = keyName;
    sound = soundFile;
    upImage = img1;
    downImage = img2;
    setImage (upImage);
    isDown = false;
}
```

**Change the Key Class so
That It Can Make Either
White or Black Keys**



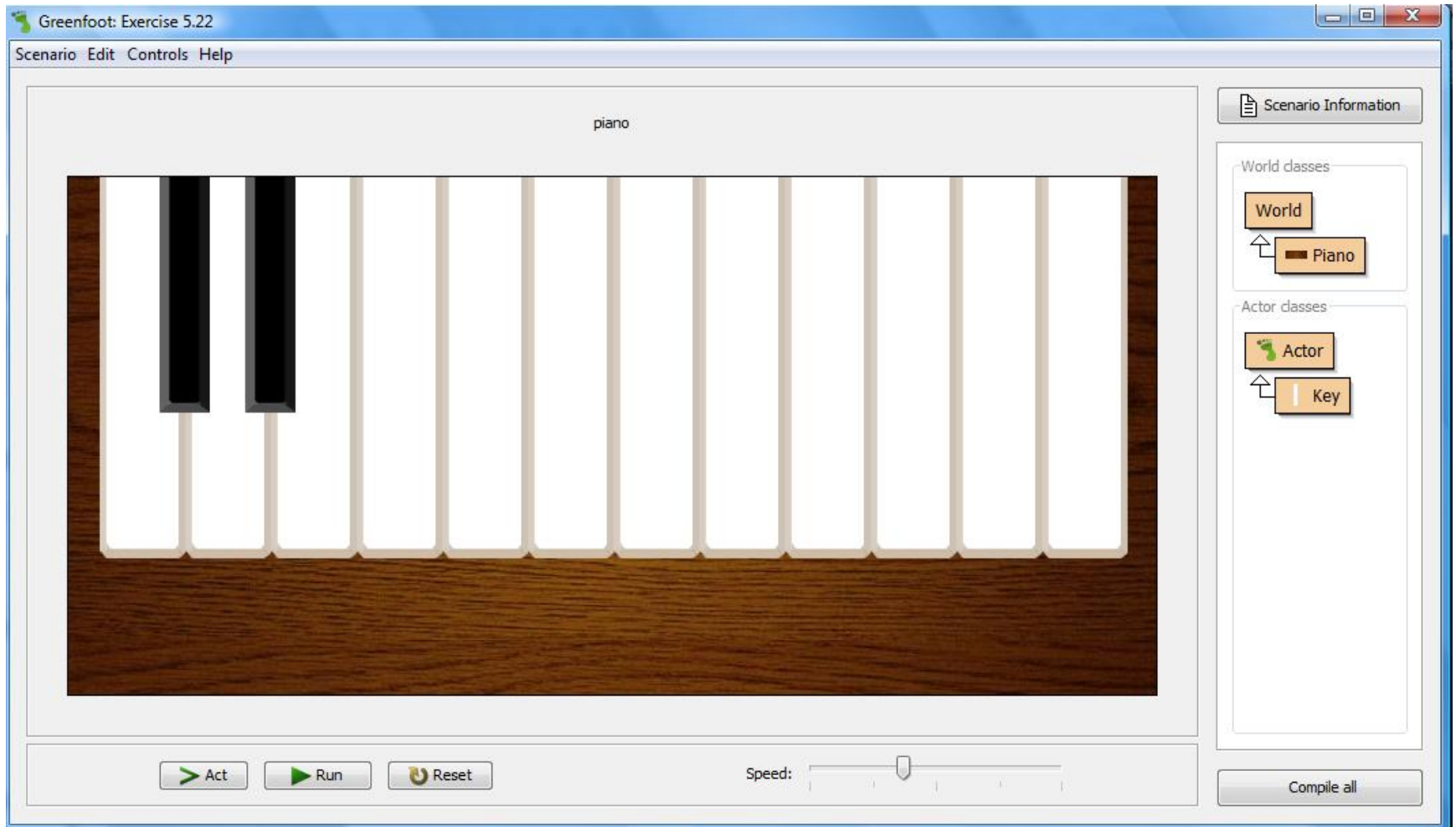
Exercise 5.22

```
public void makeKeys()
{
    int i;
    Key key;

    /*
     * Make the White Keys
     */
    for (i=0; i< whiteKeys.length; i++)
    {
        key = new Key (whiteKeys[i], whiteNotes[i] + ".wav", "white-key.png", "white-key-down.png");
        addObject (key, 54 + i*63, 140);
    }

    /*
     * Make the Black Keys
     */
    key = new Key(blackKeys[0], blackNotes[0]+".wav", "black-key.png", "black-key-down.png");
    addObject(key, 85 + (0*63), 86);
    key = new Key(blackKeys[1], blackNotes[1]+".wav", "black-key.png", "black-key-down.png");
    addObject(key, 85 + (1*63), 86);
}
```

Exercise 5.22

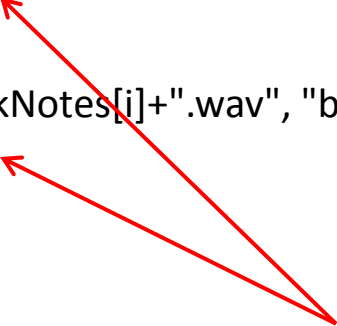


Exercise 5.23

```
/*
 * Make the Black Keys
 */
i = getValidIndex ();
key = new Key(blackKeys[i], blackNotes[i]+".wav", "black-key.png", "black-key-down.png");
addObject(key, 85 + (i*63), 86);
i = getValidIndex ();
key = new Key(blackKeys[i], blackNotes[i]+".wav", "black-key.png", "black-key-down.png");
addObject(key, 85 + (i*63), 86);
}

public int getValidIndex ()
{
    int i;

    i = Greenfoot.getRandomNumber (12);
    while ( blackKeys[i].equals("") )
        i = Greenfoot.getRandomNumber (12);
    return (i);
}
```



Add Two Black Keys

Exercise 5.23

The screenshot displays the Greenfoot IDE interface for a scenario titled "Exercise 5.23". The main workspace shows a piano keyboard with 12 keys. Two keys, the 4th and 5th from the left, are highlighted in black, indicating they are currently selected or active. The keyboard is set against a dark wood background.

The interface includes a menu bar with "Scenario", "Edit", "Controls", and "Help". A "Scenario Information" panel on the right lists the classes used in the scenario:

- World classes:**
 - World
 - Piano
- Actor classes:**
 - Actor
 - Key

At the bottom of the IDE, there are control buttons: "Act" (with a right arrow), "Run" (with a play button), and "Reset" (with a circular arrow). A "Speed" slider is also present, currently set to a low value. A "Compile all" button is located at the bottom right.

Exercise 5.24

```
private String[] whiteKeys =
    { "A", "S", "D", "F", "G", "H", "J", "K", "L", ";", "", "\\"};
private String[] whiteNotes =
    { "3c", "3d", "3e", "3f", "3g", "3a", "3b", "4c", "4d", "4e", "4f", "4g" };

private String[] blackKeys =
    { "W", "E", "", "T", "Y", "U", "", "O", "P", "", "]" };
private String[] blackNotes =
    { "3c#", "3d#", "", "3f#", "3g#", "3a#", "", "4c#", "4d#", "", "4f#" };
```

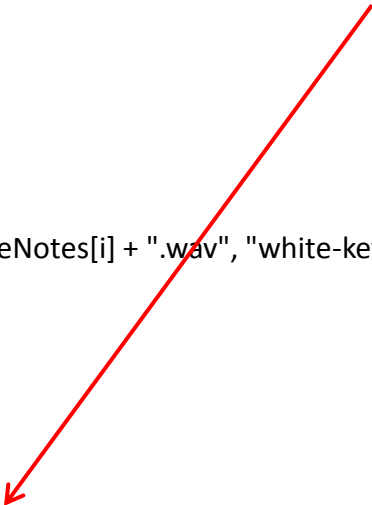
Exercise 5.25

```
public void makeKeys()
{
    int i;
    Key key;

    /*
     * Make the White Keys
     */
    for (i=0; i< whiteKeys.length; i++)
    {
        key = new Key (whiteKeys[i], whiteNotes[i] + ".wav", "white-key.png", "white-key-down.png");
        addObject (key, 54 + i*63, 140);
    }

    /*
     * Make the Black Keys
     */
    for(i = 0; i < whiteKeys.length-1; i++)
    {
        if( ! blackKeys[i].equals("") )
        {
            key = new Key(blackKeys[i], blackNotes[i]+".wav", "black-key.png", "black-key-down.png");
            addObject(key, 85 + (i*63), 86);
        }
    }
}
```

**Add Another Loop to
Create the Black Keys
This Code Must Handle the
Gaps That Exist with the
Black Keys**



Exercise 5.25

The screenshot displays the Greenfoot IDE interface for 'Exercise 5.25'. The main workspace shows a piano keyboard with 12 keys. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar with 'Act', 'Run', and 'Reset' buttons, and a 'Speed' slider. On the right, the 'Scenario Information' panel lists 'World classes' (World, Piano) and 'Actor classes' (Actor, Key).

Greenfoot: Exercise 5.25

Scenario Edit Controls Help

piano

World classes

- World
- Piano

Actor classes

- Actor
- Key

Act Run Reset

Speed: [Slider]

Compile all

Exercise 5.26

```
/*  
 * Display a message  
 */  
public void showMessage()  
{  
    GreenfootImage bg = getBackground();  
    bg.setColor (Color.WHITE);  
    bg.drawString ("Click Run and then use your keyboard to play", 25, 320);  
}
```


get and set Background

getBackground

```
public Color getBackground()
```

Gets the background color of this component.

Returns:

this component's background color; if this component does not have a background color, the background color of its parent is returned

Since:

JDK1.0

See Also:

[setBackground\(java.awt.Color\)](#)

setBackground

```
public void setBackground(Color c)
```

Sets the background color of this component.

The background color affects each component differently and the parts of the component that are affected by the background color may differ between operating systems.

Parameters:

c - the color to become this component's color; if this parameter is null, then this component will inherit the background color of its parent

Since:

JDK1.0

See Also:

[getBackground\(\)](#)

get and set Color

getColor

```
public java.awt.Color getColor()
```

Return the current drawing color.

Returns:

The current color.

setColor

```
public void setColor(java.awt.Color color)
```

Set the current drawing color. This color will be used for subsequent drawing operations.

Parameters:

color - The color to be used.

drawString

drawString

```
public void drawString(java.lang.String string,  
                       int x,  
                       int y)
```

Draw the text given by the specified string, using the current font and color. The baseline of the leftmost character is at position (x, y).

Parameters:

string - the string to be drawn.
x - the x coordinate.
y - the y coordinate.

java.awt.Color

A Color object represents a color. Color class provides static fields that return a specific Color object: BLACK, BLUE, GREEN, RED, CYAN, ORANGE, YELLOW.

For example, to obtain a Color object that represents green, use this code:

```
Color color = Color.GREEN;
```

Create a custom color by passing red-green-blue (RGB) values to the Color class's constructor:

```
Color myColor = new Color (246, 27, 27);
```

To change a component's color, call the setForeground and setBackground methods of the component.

```
component.setForeground (Color.YELLOW);
```

```
component.setBackground (Color.RED);
```

Abstract Window Toolkit (AWT)

The **Abstract Window Toolkit** (AWT) is [Java](#)'s original platform-dependent [windowing](#), [graphics](#), and [user-interface widget toolkit](#). The AWT is now part of the [Java Foundation Classes](#) (JFC) — the standard [API](#) for providing a [graphical user interface](#) (GUI) for a Java program.

AWT Architecture

The **AWT** provides two levels of [APIs](#):

- A general interface between Java and the native system, used for [windowing](#), [events](#), [layout managers](#). This API is at the core of Java [GUI](#) programming and is also used by [Swing](#) and [Java 2D](#). It contains:
 - The interface between the native windowing system and the Java application;
 - The core of the GUI [event](#) subsystem;
 - Several [layout managers](#);
 - The interface to [input devices](#) such as [mouse](#) and [keyboard](#); and
 - A [java.awt.datatransfer package](#) for use with the [Clipboard](#) and [Drag and Drop](#).
- A basic set of GUI widgets such as buttons, text boxes, and menus. It also provides the [AWT Native Interface](#), which enables [rendering libraries](#) compiled to [native code](#) to draw directly to an AWT [Canvas object](#) drawing surface.

Exercise 5.26

The screenshot displays the Greenfoot IDE interface for 'Exercise 5.26'. The main workspace shows a piano keyboard with the word 'piano' centered above it. Below the keyboard, a text prompt reads: 'Click Run and then use your keyboard to play'. The right-hand sidebar contains a 'Scenario Information' panel with two sections: 'World classes' and 'Actor classes'. Under 'World classes', there is a 'World' class box and a 'Piano' class box with an inheritance arrow pointing from Piano to World. Under 'Actor classes', there is an 'Actor' class box and a 'Key' class box with an inheritance arrow pointing from Key to Actor. The bottom control bar includes buttons for 'Act', 'Run', and 'Reset', a 'Speed' slider, and a 'Compile all' button.

5.26 Summary of Programming Techniques

In this chapter, we have seen two very fundamental and important concepts for more sophisticated programming: loops and arrays.

Loops allow us to write code that executes a sequence of statements many times over.

The other major new concept we used was an array. An array can provide many variables (all of the same type) in one single object.

Concept Summary

Concept summary

- **Logic operators**, such as && (AND) and ! (NOT), can be used to combine multiple boolean expressions into one boolean expression.
- **Abstraction** occurs in many different forms in programming. One of them is the technique to write code that can solve a whole class of problems, rather than a single specific problem.
- A **loop** is a statement in programming languages that can execute a section of code multiple times.
- A **local variable** is a variable that is declared inside a method body. It is used for temporary storage.
- An **array** is an object that holds multiple variables. These can be accessed using an **index**.
- Individual **elements** in an array are accessed using square brackets ([]) and an index to specify the array element.
- The plus symbol (+), when used with Strings, stands for **String concatenation**. It merges two Strings together into one.
- The type **String** is defined by a normal class. It has many useful methods, which we can look up in the Java library documentation.