

> Lesson Summary and Outcomes

In this lesson, students will learn to generate animation in applets. The animation will run when the user presses a key. To read and interpret keys, students will learn to implement the `KeyListener` interface. They will also learn how to capture specific keys and use them to determine the direction of the animation. For example, in the accompanying version of the code, pressing the letter 'D' will cause the object to move right. Additional keys could be coded to implement movements in other directions.

> Student Assessment

How well do students perform each of the following tasks:

- Complete the `Figure` class
- Code and compile the `Keys` class
- Modify the `Figure` class to allow for moving left, up, and down
- Add descriptive text

> This Lesson Targets the Following Subject Areas(s):

Pre-Java	Java Programming
<input type="checkbox"/> Hardware Basics	<input checked="" type="checkbox"/> Applet Programming
<input checked="" type="checkbox"/> Software Basics	<input type="checkbox"/> Subroutine Programming
<input type="checkbox"/> Networks and Servers	<input type="checkbox"/> Full-Scale Programming
<input type="checkbox"/> HTML	
<input type="checkbox"/> Action Scripting	
<input type="checkbox"/> Java Scripting	

> Prerequisite Knowledge and Skills

BlueJ:

- Creating projects
- Adding new classes
- Running applets

Java:

- Understanding the methods required for applets
- Using the `KeyListener` interface
- Using the `KeyEvent` methods

> Curriculum-Framing Questions

Essential Questions

- How does the `KeyListener` interface work?
- What makes the `KeyListener` interface useful?

Activity Questions

- What are `KeyEvents`?
- How does the program determine which key has been pressed?

Sample Content Questions

- How does the stick figure move?
- Which methods will you add to the `Figure` class?
- How does the animation work?
- How do you position components?

in an applet? Which methods of Time have now been replaced by corresponding Calendar methods and why?

- What methods of the Runnable interface must be implemented?

> Targeted Content Standards, Benchmarks, or State Frameworks

Colorado State Standards:

Mathematics:

- **Standard 4** – Students use geometric concepts, properties, and relationships in problem-solving situations and communicate the reasoning used in solving these problems. In order to meet this standard, a student will connect various physical objects with their geometric representation. They are also able to recognize, draw, describe, and analyze geometric shapes in one, two, and three dimensions.
 - **Grades 9–12** – As students in grades 9–12 extend their knowledge, what they know and are able to do includes finding and analyzing relationships among geometric figures using

International Society of Technology Education (ISTE):

Mathematics Standards:

- **Standard 3 (Geometry and Spatial Sense)** – Mathematics instructional programs should include attention to geometry and spatial sense so that all students can:
 - Analyze characteristics and properties of two- and three-dimensional geometric objects
 - Select and use different representational systems, including coordinate geometry and graph theory

transformations (for example, reflections, translations, rotations, dilations) in coordinate systems.

- Recognize the usefulness of transformations and symmetry in analyzing mathematical situations
- Use visualization and spatial reasoning to solve problems both within and outside of mathematics

> Materials and Resources Required for Activity

Equipment	<ul style="list-style-type: none"> • Computer for each student with <i>BlueJ</i> and Java SDK installed
Consumable Supplies	<ul style="list-style-type: none"> • Diskettes • Printed output for prototype
Textbooks/Lesson Guides	<ul style="list-style-type: none"> • Any of the better high-school level Java textbooks • Handouts of lessons
Technology	<ul style="list-style-type: none"> • Windows-based machines running Windows 98 SE or higher
Internet Resources	<ul style="list-style-type: none"> • http://java.sun.com/j2se/1.5.0/docs/api/

> Activity Outline

In this activity, students will learn about generating simple animation using a series of small movements. They will also learn how to implement the `KeyListener` interface and how to decode `KeyEvent` codes.

Begin by creating a project called `Animation` and within that project create a class called `Keys`. What follows here is an outline of the final `Keys` code. Write, but do not yet compile, the code, as this will require the addition of the `Figure` class.

```
/**
 * Keys.java is an intro to getting keyboard input in an applet
 *
 * @author (your name)
 * @version (a version number or a date)
 */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Keys extends Applet implements KeyListener
{
    // define instance variables
    Figure stick; // stick figure
    public void init()
    {
        // Initialize drawing colors
        setBackground(Color.white);
        setForeground(Color.black);
        // create the Figure object by sending the window size
        stick = new Figure(this.getWidth(), this.getHeight());
        addKeyListener(this); // register the KeyListener with
```

```

        // the applet
    } // end of init() method
    public void paint(Graphics g)
    {
        stick.draw(g); // draw the figure, that's all
    } // end of paint() method
    public void keyPressed(KeyEvent event)
    {
        int keyCode = event.getKeyCode(); // get the key hit
        if (keyCode == KeyEvent.VK_D)
        {
            stick.moveRight();
            repaint(); // signal Java to redraw the window
        }
        event.consume(); // keep anything else from using
                        // the keyrepaint();
    }
    public void keyReleased(KeyEvent event)
    {
        event.consume(); // ignore keyReleased
    }
    public void keyTyped(KeyEvent event)
    {
        event.consume(); // ignore keyTyped
    }
} // end of Keys class

```

To make this project work, the applet will need to implement the `KeyListener` interface. You can do this in the class declaration where it says:

```
implements KeyListener
```

Next, to make the `KeyListener` work in conjunction with this application, use code containing the line:

```
add(KeyListener(this));
```

What this means is that when running, the program will continue to “listen,” or wait, for key presses. When you implement the `KeyListener` interface, ensure that the code defines three methods:

```
public void keyPressed(KeyEvent e);
```

```
public void keyReleased(KeyEvent e);
public void keyTyped(KeyEvent e);
```

Implement the `KeyPressed()` routine to capture and evaluate the key that has been pressed. If that key was a d (capital or lowercase), the stick figure will move to the right. The other methods, `keyReleased()` and `keyTyped()`, will simply call `KeyEvent`'s `consume` method and discard any actions. Within the `keyPressed()` method, the application will retrieve the code corresponding to the key pressed.

```
int keyCode = event.getKeyCode(); // get the key hit
```

Implement the `KeyPressed()` routine to capture and evaluate the key that has been pressed. If that key was a d (capital or lowercase), the stick figure will move to the right. The other methods, `keyReleased()` and `keyTyped()`, will simply call `KeyEvent`'s `consume` method and discard any actions. Within the `keyPressed()` method, the application will retrieve the code corresponding to the key pressed.

```
int keyCode = event.getKeyCode(); // get the key hit
```

Each key has a corresponding `keyCode`. Here is a table indicating keys and their respective codes.

Letters:	VK_A to VK_Z
Numbers:	VK_0 to VK_9
Arrows:	VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN
Number pad:	VK_NUMPAD0 to VK_NUMPAD9
Others:	VK_NUMPAD0 to VK_NUMPAD9

To make this application work, you will need to complete the `Figure` class. This is the class from which `stickFigure` is instantiated. The skeleton of that class is given as follows:

```
import java.awt.*;

import java.awt.geom.*;

/**
 * A figure that can be manipulated and that draws itself on a
 * canvas.
 *
 * @author
 * @version 1.0
 */
public class Figure
{
    private int xPosition;
    private int yPosition;
    private int width; // of window

    private int height; // of window
    /**
     * Create a new circle at default position with default color.
     */
    public Figure(int windowWidth, int windowHeight)
    {
    }
    /**
     * Move the figure a few pixels to the right.
     */
    public void moveRight()
    {
    }
    /**
     * Draw the circle with current specifications on screen.
     */
    public void draw(Graphics g)
    {
    }
}
```

Implement the constructor to calculate the center of the window. Then, using the `draw()` method, create a simple figure positioned in that center. The `moveRight()` method will then reposition the figure 20 pixels from its origin and redraw it.

Next, complete the `Figure` class. It should include a constructor that stores window size and sets the starting position to its center. Add a `moveRight()` method to move the coordinates to the right by some default amount, compile it, and then compile and run `Keys`.

Finally, add enhancements to the applet.

Tips, Hints, and Tricks: Add movement to the left, up, and down. Convert to the cursor movement keys if desired. These are the four keys usually located to the right of the alphabetical portion of the keyboard and include arrows pointing up, down, left and right.

> Pacing/Timeline

- Code `keys` class: 1 day
- Define and implement new `Figure` class: 1 day
- Compile, debug, and run the project: 1 day
- Add other movement options (left, up, down): 1–2 days

> Teacher Reflection (For example, what worked well in this activity? What would you change if you were to teach it again?)

- Were students able to define and display the basic stick figure?
- Were they able to add new movements: left, up, down?
- Did they try other key combinations, such as the cursor movement keys?

> Student Handout: Java Programming Animation and Key Listeners

In this unit, students will learn about simple animation by using a series of small movements. They will also learn how to implement the `KeyListener` interface and how to decode `KeyEvent` codes.

Begin by creating a project called Animation and within that project create a class called `Keys`. What follows here is an outline of the final `Keys` code. Write, but do not yet compile, the code, as this will require the addition of the `Figure` class.



```
/**
 * Keys.java is an intro to getting keyboard input in an applet
 *
 * @author (your name)
 * @version (a version number or a date)
 */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Keys extends Applet implements KeyListener
{
    // define instance variables
    Figure stick;
    public void init()
    {
        // Initialize drawing colors
        setBackground(Color.white);
        setForeground(Color.black);
        // create the Figure object by sending the window size
        stick = new Figure(this.getWidth(), this.getHeight());
        addKeyListener(this); // register the KeyListener with
```

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```

                                //the applet
    } // end of init() method

    public void paint(Graphics g)
    {
        stick.draw(g); // draw the figure, that's all
    } // end of paint() method
    public void keyPressed(KeyEvent event)
    {
        int keyCode = event.getKeyCode(); // get the key hit
        if (keyCode == KeyEvent.VK_D)
        {
            stick.moveRight();
            repaint();
        }
        event.consume(); // keep anything else from using the
        keyrepaint();
        // signal Java to redraw the window
    }
    public void keyReleased(KeyEvent event)
    {
        event.consume(); // ignore keyReleased
    }
    public void keyTyped(KeyEvent event)
    {
        event.consume(); // ignore keyTyped
    }
} // end of Keys class
```

To make this project work, the applet will need to implement the `KeyListener` interface. You can do this in the class declaration where it says

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```
implements KeyListener
```

Next, to make the `KeyListener` work in conjunction with this application, use code containing the line

```
add(KeyListener(this));
```

What this means is that when running, the program will continue to “listen,” or wait, for key presses. When you implement the `KeyListener` interface, ensure that the code defines three methods:

```
public void keyPressed(KeyEvent e);  
public void keyReleased(KeyEvent e);  
public void keyTyped(KeyEvent e);
```

Implement the `KeyPressed` routine to capture and evaluate the key that has been pressed. If that key was a `d` (capital or lowercase), the stick figure will move to the right. The other methods, `keyReleased()` and `keyTyped()`, will simply call `KeyEvent`'s `consume()` method and discard any actions. Within the `keyPressed()` method, the application will retrieve the code corresponding to the key pressed.

```
int keyCode = event.getKeyCode(); // get the key hit
```

Each key has a corresponding `keyCode`. Here is a table indicating keys and their respective codes.

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

Letters:	VK_A to VK_Z
Numbers:	VK_0 to VK_9
Arrows:	VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN
Number pad:	VK_NUMPAD0 to VK_NUMPAD9
Others:	VK_SPACE, VK_TAB, VK_ALT, VK_CTRL

To make this application work, you will need to complete the `Figure` class. This is the class from which `stickFigure` is instantiated. The skeleton of that class is given as follows:

```
import java.awt.*;
import java.awt.geom.*;
/**
 * A figure that can be manipulated and that draws itself on a
 * canvas.
 *
 * @author
 * @version 1.0
 */
public class Figure
{
    private int xPosition;
    private int yPosition;
    private int width; // of window
    private int height; // of window
```

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```
/**
 * Create a new circle at default position with default
 * color.
 */
public Figure(int windowWidth, int windowHeight)
{
}
/**
 * Move the figure a few pixels to the right.
 */
public void moveRight()
{
}
/*
 * Draw the circle with current specifications on screen.
 */
public void draw(Graphics g)
{
}
}
```

> YOUR JOB:

Complete the `Figure` class, implementing the constructor and the `moveRight()` method

Compile both classes, debug, run, and test the project

> EXTRAS YOU CAN ADD:

Add methods to `moveLeft()`, `moveUp()`, and `moveDown()`