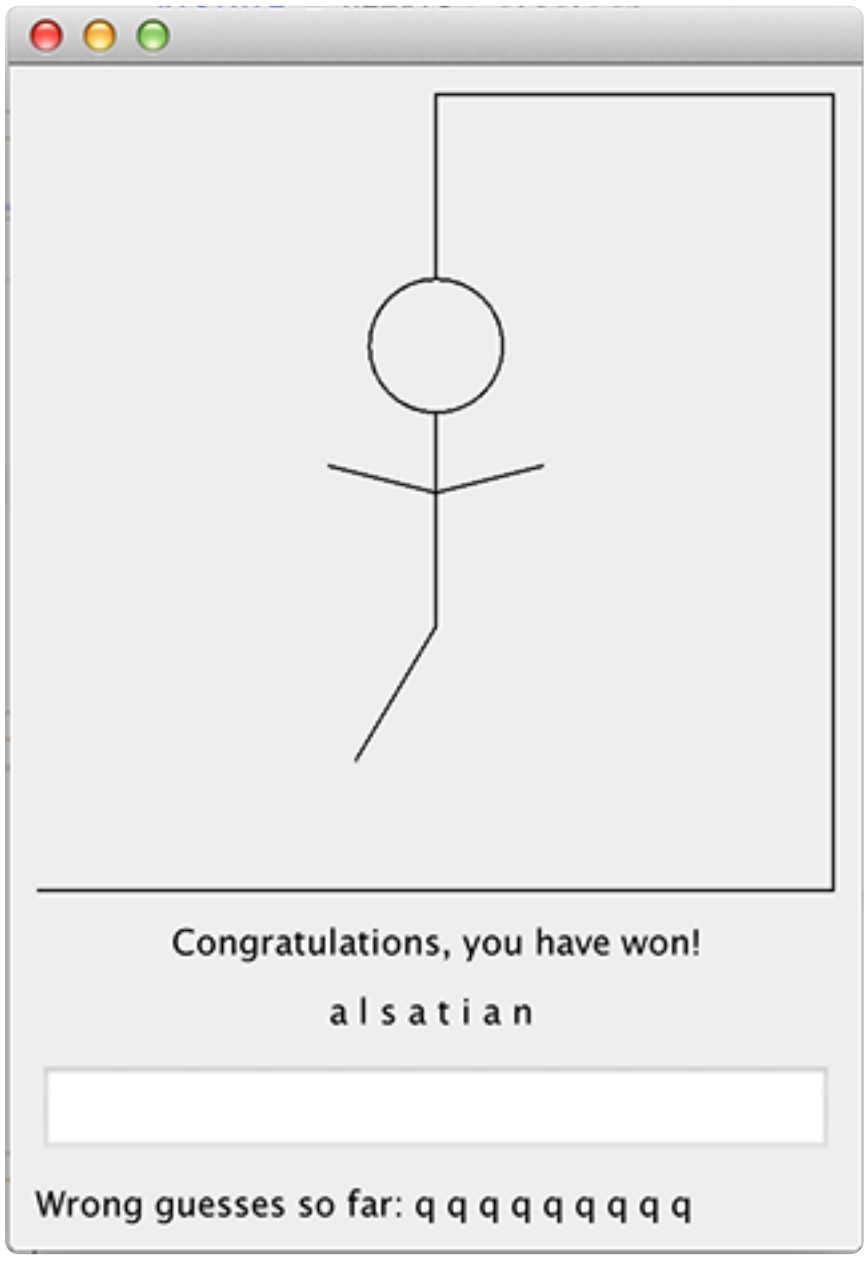
# Java GUI Hangman Application

## The final result




Create a new project in and start by creating a new class file called *MainWindow.java*. This class will represent the view of our application.

```
public class MainWindow extends JFrame {

}
```

This class extends JFrame. This means that it will be inheriting all the methods and fields of JFrame, which essentially is a window with close, minimize and maximize buttons.

Now we create a few variables.

```
public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

}
```

The variables are self-explanatory. Nevertheless, here are the details:

- **remainingGuesses** represents an integer which is the number of guesses remaining.
- **wrongGuesses** represents a String containing all the wrong guesses been made so far.
- **word** represents the word the user is guessing
- **visible** represents the String that is shown to the user (eg: if the word was *cat*, visible would be _ _ _).

Now we create a constructor and initialise a few variables.

```
public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;
    }
}
```

- Our application of hangman allowed a total of 10 guesses to the user, therefore *remainingGuesses* is initialised to 10.
- *wrongGuesses* is initially an empty String.
- We are passing the word we are guessing (eg: *cat*) through the constructor as the variable *toGuess*, therefore *word* is initialised to the value of *toGuess*.

Now that we know the word that the user will be guessing, we need to assign a value to the *visible* variable. So remember, if the word is *cat*, the value of visible will be _ _ _.

```
public class MainWindow extends JFrame {

    private int remainingGuesses;
```

```
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";

        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }
    }
}
```

Firstly, we have to initialise *visible*, which we did assigning it an empty String (*visible = ""*). Then we need to add underscores and spaces ("_ ") to it for each character in the word. We can accomplish this using a for loop. So for example if the word is *cat*, the loop will be repeated 3 times. So the value of *visible* will "_ _ _ ".

Now that we have initialised all the important variables, we can move onto the GUI part. We are going to be using *BorderLayout* a lot in this application. So it's very important you understand it.

JFrame and JPanel can have layouts. BorderLayout essentially separates the panel or frame into 5 parts. North, South, East, West, and Centre. **Note**: these parts will have *0 width* when there aren't any components added to them. JPanel, JLabel, JTextField, JButton, and many more are examples of a few components.

So first, we make a centralised JPanel which holds all the components and add it to the JFrame. Remember: JFrame by default has its layout set to *BorderLayout*.

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";

        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }

        JPanel corePanel = new JPanel();
        corePanel.setLayout(new BorderLayout());
```

```
        this.add(corePanel, BorderLayout.CENTER);
    }
}
```

*corePanel* is a new JPanel which is added to the *center part* of the JFrame. The layout of corePanel is also *BorderLayout*. The file imports are added automatically on *Eclipse*.

Now let's display the window, so that we can see what is actually happening.

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";
        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }

        JPanel corePanel = new JPanel();
        corePanel.setLayout(new BorderLayout());

        this.add(corePanel, BorderLayout.CENTER);

        this.pack();
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new MainWindow("cat");
    }
}
```

- *pack()* is to make its size big enough to fit all the components
- *setLocationRelativeTo()* is to make the window appear in the center of the screen
- *setDefaultCloseOperation()* is to make the application close completely when the user clicks on the X.
- *setVisible()* is to make the window that we are making visible.
- *new MainWindow()* is to actually create a new window. Remember Java applications always start from the static main method.

Now let's create some JLabels and a JTextField to display texts and an input box.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";

        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }

        JPanel corePanel = new JPanel();
        corePanel.setLayout(new BorderLayout());

        JLabel status = new JLabel("You have "+remainingGuesses+" remaining", SwingConstants.CENT
ER);
        JLabel wrong = new JLabel("Wrong guesses so far: "+wrongGuesses);
        JLabel visibleLabel = new JLabel(visible, SwingConstants.CENTER);
        JTextField input = new JTextField();

        JPanel southPanel = new JPanel(new GridLayout(4, 1));
        southPanel.add(status);
        southPanel.add(visibleLabel);
        southPanel.add(input);
        southPanel.add(wrong);

        corePanel.add(southPanel, BorderLayout.SOUTH);

        this.add(corePanel, BorderLayout.CENTER);

        this.pack();
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new MainWindow("cat");
    }
}
```

- *SwingConstants.CENTER* is used to align it to the center. Default alignment is left.
- *GridLayout* represents a layout with 4 rows and 1 column. Equally sized.

The components are added to the southPanel's GridLayout's rows in order. Then, the southPanel is added to the corePanel's south component. The reason for this is that the center component of corePanel will be taken by the hangman figure.

Now comes the second best part, the hangman figure. Before that, you must understand a few basic functionalities.

- Every JPanel has a method called *paintComponent()*. This method is used to add graphics to the JPanel.
- *drawLine(fromX, fromY, toX, toY)* method draws a line from the co-ordinates mentioned to the co-ordinates mentioned.
- *drawOval(fromX, fromY, width, height)* method draws an oval (circle) from the co-ordinates mentioned.

Now let's create a new class called *HangmanFigure.java*.

```
import javax.swing.JPanel;

public class HangmanFigure extends JPanel {

}
```

Now let's create necessary variables. Hint: our figure is going to depend on the number of guesses the user has remaining.

```
import javax.swing.JPanel;

public class HangmanFigure extends JPanel {

    private int guesses;

}
```

Now let's create a constructor.

```
import javax.swing.JPanel;
import java.awt.Dimension;

public class HangmanFigure extends JPanel {

    private int guesses;

    public HangmanFigure() {
        super();
        guesses = 0;
        setPreferredSize(new Dimension(300, 300));
        setOpaque(true);
    }

}
```

- *super()* calls the constructor of its parent (i.e.: JPanel). This must be called in order to retain the normal functionalities of our class.
- *guesses* is initialised to 0 as user has made 0 guesses to start with.
- *setPreferredSize()* sets the size of the panel. 300x300 pixels seems to be optimum.
- *setOpaque(true)* makes sure that everything in the background in invisible to the user.

Now let's set the base for some graphics. Remember the *paintComponent* method I mentioned in JPanels.

```
import javax.swing.JPanel;
import java.awt.Dimension;
import java.awt.Graphics;

public class HangmanFigure extends JPanel {

    private int guesses;

    public HangmanFigure() {
        super();
        guesses = 10;
        setPreferredSize(new Dimension(300, 300));
        setOpaque(true);
    }

    public void paintComponent(Graphics g) {

    }
}
```

So basically, whatever graphics I create in this method will be added to the panel.

```
import java.awt.Color;
import javax.swing.JPanel;
import java.awt.Dimension;
import java.awt.Graphics;

public class HangmanFigure extends JPanel {

    private int guesses;

    public HangmanFigure() {
        super();
        guesses = 10;
        setPreferredSize(new Dimension(300, 300));
        setOpaque(true);
    }

    public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);

        // base
        if(guesses > 0) {
            g.drawLine(1, 299, 299, 299);
        }
```

```
    }

}
```

- *setColor()* method sets the colour of the line/oval you are drawing using Java.
- So if guess > 0, draw a line from co-ordinates 1, 299 (i.e.: the bottom left hand corner of the panel) to the co-ordinates 299, 299 (i.e.: the bottom right hand corner of the panel)

Let's continue:

```
import java.awt.Color;
import javax.swing.JPanel;
import java.awt.Dimension;
import java.awt.Graphics;

public class HangmanFigure extends JPanel {

    private int guesses;

    public HangmanFigure() {
        super();
        guesses = 10;
        setPreferredSize(new Dimension(300, 300));
        setOpaque(true);
    }

    public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);

        // base
        if(guesses > 0) {
            g.drawLine(1, 299, 299, 299);
        }

        // right wall
        if(guesses > 1) {
            g.drawLine(299, 299, 299, 1);
        }

        // top line
        if(guesses > 2) {
            g.drawLine(150, 1, 299, 1);
        }

        // hanging line
        if(guesses > 3) {
            g.drawLine(150, 1, 150, 70);
        }

        // face
        if(guesses > 4) {
            g.drawOval(125, 70, 50, 50);
        }

        // body
        if(guesses > 5) {
            g.drawLine(150, 120, 150, 200);
```

```
        }

        // left hand
        if(guesses > 6) {
            g.drawLine(150, 150, 110, 140);
        }

        // right hand
        if(guesses > 7) {
            g.drawLine(150, 150, 190, 140);
        }

        // left leg
        if(guesses > 8) {
            g.drawLine(150, 200, 120, 250);
        }

        // right leg
        if(guesses > 9) {
            g.drawLine(150, 200, 180, 250);
        }
    }

}
```

- *right wall* - draw a line from the co-ordinates 299, 299 (i.e.: the bottom right hand corner of the panel) to the co-ordinates 299, 1 (i.e.: the top right hand corner of the panel)
- *top line* - draw a line from the co-ordinates 150, 1 (i.e.: the highest mid-point of the panel) to the co-ordinates 299, 1 (i.e.: the top right hand corner of the panel)
- *hanging line* - draw a line from the co-ordinates 150, 1 (i.e.: the highest mid-point of the panel) to the co-ordinates 150, 70 (i.ie.: a little lower mid-point of the panel)
- *face* - draw an oval from the co-ordinates 125, 70 (i.e.: a little left of where the hanging line left off) and make it 50x50 in width and height
- *body* - draw a line from the co-ordinates 150, 120 (i.e.: the end of the face) to the co-ordinates 150, 200 (i.e.: a near bottom mid-point of the panel)
- and so on...

That will produce a splendid hangman figure. Now we basically need a method which we can call everytime a user guesses wrongly so that the hangman figure gets updated.

```
import java.awt.Color;
import javax.swing.JPanel;
import java.awt.Dimension;
import java.awt.Graphics;

public class HangmanFigure extends JPanel {

    private int guesses;

    public HangmanFigure() {
        super();
        guesses = 10;
        setPreferredSize(new Dimension(300, 300));
        setOpaque(true);
```

```
    }

    public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);

        // base
        if(guesses > 0) {
            g.drawLine(1, 299, 299, 299);
        }

        // right wall
        if(guesses > 1) {
            g.drawLine(299, 299, 299, 1);
        }

        // top line
        if(guesses > 2) {
            g.drawLine(150, 1, 299, 1);
        }

        // hanging line
        if(guesses > 3) {
            g.drawLine(150, 1, 150, 70);
        }

        // face
        if(guesses > 4) {
            g.drawOval(125, 70, 50, 50);
        }

        // body
        if(guesses > 5) {
            g.drawLine(150, 120, 150, 200);
        }

        // left hand
        if(guesses > 6) {
            g.drawLine(150, 150, 110, 140);
        }

        // right hand
        if(guesses > 7) {
            g.drawLine(150, 150, 190, 140);
        }

        // left leg
        if(guesses > 8) {
            g.drawLine(150, 200, 120, 250);
        }

        // right leg
        if(guesses > 9) {
            g.drawLine(150, 200, 180, 250);
        }
    }

    public void set() {
        guesses++;
        paintComponent(this.getGraphics());
    }
```

```
}
```

This method will update the hangman figure by incrementing the guesses. *getGraphics()* is pre-built in JPanel.

Now we can add the hangman figure to our main window.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";
        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }

        JPanel corePanel = new JPanel();
        corePanel.setLayout(new BorderLayout());

        JLabel status = new JLabel("You have "+remainingGuesses+" remaining", SwingConstants.CENT
ER);
        JLabel wrong = new JLabel("Wrong guesses so far: "+wrongGuesses);
        JLabel visibleLabel = new JLabel(visible, SwingConstants.CENTER);
        JTextField input = new JTextField();

        JPanel southPanel = new JPanel(new GridLayout(4, 1));
        southPanel.add(status);
        southPanel.add(visibleLabel);
        southPanel.add(input);
        southPanel.add(wrong);

        corePanel.add(southPanel, BorderLayout.SOUTH);

        HangmanFigure hf = new HangmanFigure();
        corePanel.add(hf, BorderLayout.CENTER);

        this.add(corePanel, BorderLayout.CENTER);

        this.pack();
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
```

```
    }

    public static void main(String[] args) {
        new MainWindow("cat");
    }
}
```

We create a new hangman figure instance and add it to the center compartment of the corePanel.

That's our GUI completed! Now we can focus on what happens when a user input's a character in the input text box and presses enter. These are called *ActionListeners*. We need to create one that only executes once the user presses enter while focusing on the text box.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MainWindow extends JFrame {

    private int remainingGuesses;
    private String wrongGuesses;
    private String word;
    private String visible;

    public MainWindow(String toGuess) {
        remainingGuesses = 10;
        wrongGuesses = "";
        word = toGuess;

        visible = "";

        for(int i = 0; i < word.length(); ++i) {
            visible += "_ ";
        }

        JPanel corePanel = new JPanel();
        corePanel.setLayout(new BorderLayout());

        JLabel status = new JLabel("You have "+remainingGuesses+" remaining", SwingConstants.CENT
ER);
        JLabel wrong = new JLabel("Wrong guesses so far: "+wrongGuesses);
        JLabel visibleLabel = new JLabel(visible, SwingConstants.CENTER);
        JTextField input = new JTextField();

        JPanel southPanel = new JPanel(new GridLayout(4, 1));
        southPanel.add(status);
        southPanel.add(visibleLabel);
        southPanel.add(input);
        southPanel.add(wrong);

        corePanel.add(southPanel, BorderLayout.SOUTH);
```

```
        HangmanFigure hf = new HangmanFigure();
        corePanel.add(hf, BorderLayout.CENTER);

        this.add(corePanel, BorderLayout.CENTER);

        input.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                // TODO Auto-generated method stub

            }

        });

        this.pack();
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new MainWindow("cat");
    }
}
```

So any code written in *actionPerformed* method will be executed when the user presses enter while focusing on the textbox.

So firstly in our listener, we need to check a few things:

- The input is only 1 character
- The input is a lowercase character

```
input.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String text = input.getText();
        if(text.length() == 1 && text.matches("[a-z]")) {
            // continue here
        }
        else {
            System.out.println("Invalid input!");
        }

        input.setText("");
    }

});
```

I added a keyword *final* before the JTextField input. This is because if you want to you the text field inside an action listener, it must be declared final. *getText()* and *setText()* methods access or mutate the text of the input box.

*text.matches("[a-z]")* is using Regular Expression to make sure the input is anything in between a to z (inclusive).

Now, that we have verified the input, we should make sure the guess is correct.

```
input.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String text = input.getText();
        if(text.length() == 1 && text.matches("[a-z]")) {

            boolean guessFound = false;

            for(int i = 0; i < word.length(); ++i) {
                if(text.charAt(0) == word.charAt(i)) {
                    guessFound = true;
                }
            }

            if(!guessFound) {
                // code to execute if the guess if not found
            }

        }
        else {
            System.out.println("Invalid input!");
        }

        input.setText("");
    }

});
```

- The for loop loops through every character in the word and compares it to the character the user has input, if it matches, *guessFound* is set to true. Otherwise, *guessFound* will remain false.

Now we need to decide what happens if the guess is not found. So if the user has no more guesses remaining, the program should say you have lose, the word was "cat". It should also disable the text box from capturing anymore inputs.

```
input.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String text = input.getText();
        if(text.length() == 1 && text.matches("[a-z]")) {

            boolean guessFound = false;

            for(int i = 0; i < word.length(); ++i) {
                if(text.charAt(0) == word.charAt(i)) {
                    guessFound = true;
                }
            }
```

```
            if(!guessFound) {
                if(--remainingGuesses >= 0) {
                    status.setText("You have "+remainingGuesses+" guesses remaining");
                    wrongGuesses += text+" ";
                    wrong.setText("Wrong guesses so far: "+wrongGuesses);
                    hf.set();
                }
                else {
                    status.setText("You lost: the word was "+word);
                    input.setEnabled(false);
                }
            }

        }
        else {
            System.out.println("Invalid input!");
        }

        input.setText("");
    }

});
```

- *--remainingGuesses* decrements the integer value by one. So if the decremented value is greater than or equal to 0, show the user the number of guesses remaining.
- *setEnabled(false)* disables the the text box
- *hf.set()* calls the set method we created in the hangman figure. This updates the picture of the hangman on the screen.
- JLabels status, wrong, and HangmanFigure hf, were made final inorder to use them in the action listener.

Now we need to do the other side of the story. So what if the guess if found. Firstly, when the guess is found, we need to update the value of *visible* (i.e.: replace the underscores with the letters that have been found in those positions).

```
input.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String text = input.getText();
        if(text.length() == 1 && text.matches("[a-z]")) {

            boolean guessFound = false;

            for(int i = 0; i < word.length(); ++i) {
                if(text.charAt(0) == word.charAt(i)) {
                    guessFound = true;

                    String newVisible = "";
                    for(int j = 0; j < visible.length(); ++j) {
                        if(j == (i*2)) {
                            newVisible += word.charAt(i);
                        }
                        else {
                            newVisible += visible.charAt(j);
                        }
```

```
                    }
                    visible = newVisible;
                    visibleLabel.setText(visible);
                }
            }

            if(!guessFound) {
                if(--remainingGuesses >= 0) {
                    status.setText("You have "+remainingGuesses+" guesses remaining");
                    wrongGuesses += text+" ";
                    wrong.setText("Wrong guesses so far: "+wrongGuesses);
                    hf.set();
                }
                else {
                    status.setText("You lost: the word was "+word);
                    input.setEnabled(false);
                }
            }

        }
        else {
            System.out.println("Invalid input!");
        }

        input.setText("");
    }

});
```

- *visibleLabel* is made final in order to use it in the listener
- Remember the loop is only executed if the guess found.
- First, we create a new String that will contain the updated value of *visible*. To explain the loop, assume the word is "cat" and the user has guesses "a".
- The first for loop is going through all the characters in *word*.
- It goes to the second for loop when i = 1. "cat" - the second character is "a".
- The second for loop is going through all the characters of *visible* whose initial value is: "_ _ _ ".
- Remember one character in *word* represents two characters in *visible*. This is because "cat" was shown as "_ _ _ " (underscore and space for each character).
- It checks if the current position in visible is 2 times the value of i.
- If it is, take the character from *word* and add it to *newVisible*.
- Else, take the character from the *visible* and add it to *newVisible*.
- Finally after the loop is finished, *visible* is updated to *newVisible* and the label is also updated.
- This will show "_ a _ " on the screen instead of "_ _ _ ".

Now that we have taken care of updating the main text when the guess if found, let's focus on the last bit. So if the guess is found and the whole word is guessed correctly, show the user that they have won.

```
input.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String text = input.getText();
        if(text.length() == 1 && text.matches("[a-z]")) {
```

```
        boolean guessFound = false;

        for(int i = 0; i < word.length(); ++i) {
            if(text.charAt(0) == word.charAt(i)) {
                guessFound = true;

                String newVisible = "";
                for(int j = 0; j < visible.length(); ++j) {
                    if(j == (i*2)) {
                        newVisible += word.charAt(i);
                    }
                    else {
                        newVisible += visible.charAt(j);
                    }
                }
                visible = newVisible;
                visibleLabel.setText(visible);
            }
        }

        if(!guessFound) {
            if(--remainingGuesses >= 0) {
                status.setText("You have "+remainingGuesses+" guesses remaining");
                wrongGuesses += text+" ";
                wrong.setText("Wrong guesses so far: "+wrongGuesses);
                hf.set();
            }
            else {
                status.setText("You lost: the word was "+word);
                input.setEnabled(false);
            }
        }
        else {
            String actualVisible = "";
            for(int i = 0; i < visible.length(); i+=2) {
                actualVisible += visible.charAt(i);
            }

            if(actualVisible.equals(word)) {
                status.setText("Congratulations, you have won!");
                input.setEnabled(false);
            }
        }

    }
    else {
        System.out.println("Invalid input!");
    }

    input.setText("");
}

});
```

- So we need to compare the *visible* with the word. But because visible contains all those spaces. We need to remove them and compare them after. Now the spaces take place after every character.
- That's why the for loop for *actualVisible* is looping through every 2nd character in visible.

- Once we have extracted the word out of *visible*, we compare it to the actual word that the user is trying to guess. If it is equal, the user has guessed it correctly. And therefore have won the game.

And that's it! You can set your own word through *new MainWindow("cat")*.