actions. For example, if a user can drag a product into a shopping cart, pressing the Enter key on an "add to cart" image or link may be one way to perform the same task. Implementing keyboard shortcuts for cut-and-paste functionality is another technique used to make a common drag-and-drop operation accessible. When possible, use the simplest method that requires the fewest keystrokes. This implies that links are preferable to Image components. Since a link is keyboard accessible and is capable of displaying an image as an icon, it accomplishes the same goals as the Image component, which may not have keyboard access defined. In addition, a redundant event can easily be assigned to the link that moves the product to the shopping cart; for example:

```
lnkAddToCart.addEventListener(MouseEvent.CLICK, addToCart);
function addToCart(e:MouseEvent): void
{
  // add to cart functionality goes here
}
```

Text

In some cases, uneditable text should be included in the tab order, to provide easy access to it for screen reader users. Alerts and error messages, license text, or any text that is dynamically updated may fall into this category. Placing this text in the tab order allows users of screen readers to tab to the component and have the current value of the text read to them. Unfortunately, the Text and Label components cannot simply be placed in the tab order by setting a tab index and tab enabling them. Flex Text and Label controls are not included in the tab order by the FocusManager class. There are two commonly used techniques for handling this situation. First, developers can simply use a TextInput component and set the `editable` property to `false`; for example:

```
// ActionScript
txtTimeLeft.editable = false; // ensure that the user cannot change
   the value of the TextInput
txtTimeLeft.tabIndex = 13; // set the tab order for this element
txtTimeLeft.tabEnabled = true;  // should be enabled by default but
   this will explicitly set it

// MXML
<s:TextInput id="txtTimeLeft" editable="false" tabIndex="13" tabEnabled ="true" />
```

The second technique is to extend the standard Label or Text component class and ensure the extended class implements the Focus Manager Component interface (IFocusManagerComponent); for example:

```
// create a custom class, FocusableLabel.as
package
{
  import spark.components.Label;
  import mx.managers.IFocusManagerComponent;

  // implement the focus manager
  public class FocusableLabel extends spark.components.Label implements
      IFocusManagerComponent
  {
    // the constructor
    public function FocusableLabel()
    {
      super();
    }
  }
}
```