Assignment 2 - Ideas

- Intro to MVC
- Actions
  - Buttons (Icons)
  - Menu
- Internationalization (I18N)
Model-View-Control

- A paradigm coming from SmallTalk (1980)
- MVC is a base for GUI development

MVC includes a number of Design Patterns:
- Protocols instead of components (interfaces)
- Weak coupling (Observer)
- Separation of Model and View
  - Model = the data
  - View = the presentation
Does MVC work?

- Model contains the data to display
  - Data may not exist, can be generated on the fly!
- View is the visual representation
- Control is the ”glue” between Model and View
  - knows how to modify the data

- ”Pure” MVC is difficult to implement
  - Some coupling necessary between C and MV
  - In the ideal world they should all be independent
How to...

1. Put your data into a Model
   - ListModel
   - TableModel

2. Link to a suitable representation (View)
   - JList
   - JTable

3. Design the presentation (View)

4. Manipulate the model (the data in it)
MVC

- Separate Logic and Presentation

- The arrow buttons receive the click and activate an ActionListener that changes THE MODEL

- The model notifies the presentation field to update the view of the model

http://www.enode.com/x/markup/tutorial/spinner.html#markup
Actions

- Some example actions:
  - New TODO
  - Edit TODO
  - Sort…
  - etc.

- All these are functions in the application, not properties of the Interface!
Actions

- Standard way is a top-down approach
  - Add Components to an interface and
  - tell them what they should do
    - Using ActionListeners

- Using Actions is more of a Bottom-Up approach
  - Define the functionality
  - Connect the functionality to an Action
  - Connect Actions to GUI items
Actions

- Simpler structure
- Only one thing in one place
- Reducing redundancy
- Same Action for many widgets
- Some Containers know about Actions
  - E.g. menus, tool bars (see Action docs)
javax.swing.Action

- Interface, contains
  - Accelerator, Mnemonic, Name, Icon, Description (short+long), Enabled?, command

javax.swing.AbstractAction
- default implementations for the Action interface
- Cf. Adapter classes

- new JButton(new ExitAction());
- new JButton(new ExitAction(initValue));
Action vs. ActionListener?

- ActionListeners are less complex
- Actions provide more programmer support
  - enabling, disabling
  - multiple controls
  - widget control
- Actions take up more space
- Actions are in some respect more elegant
The terms foobar, foo, bar, and baz are common placeholder names (also referred to as metasyntactic variables) used in computer programming or computer-related documentation.

They are commonly used to represent unknown values, typically when describing a scenario where the purpose of the unknown values is understood, but their precise values are arbitrary and unimportant.
ActionListener (AL)

// Not OOP
class Foo implements ActionListener {

    public Foo() {
        JButton b = new JButton();
        b.addActionListener(this);  // ugly!

        public void actionPerformed(ActionEvent e) {
            // doit
        }
    }
}
// Sometimes ok, mostly not

class Foo {
    class Bar implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            // doit
        }
    }
}

Foo() {
    JButton b = new JButton();
    b.addActionListener(new Bar());
}

class Foo {
    class Bar implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            // doit
        }
    }

    Foo() {
        Bar bar = new Bar();
        JButton b1 = new JButton();
        JButton b2 = new JButton();
        b1.addActionListener(bar);  // smart, shares action
        b2.addActionListener(bar);
    }
}
AL 3

// Good! Using anonymous classes

class Foo {

    Foo() {

        JButton b = new JButton();
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // doit
            }
        });
    }
}

class MyAction extends AbstractAction {
    ImageIcon icon = new ImageIcon(...);
    //TODO
    MyAction(String name) {
        putValue(Action.NAME, name);
        putValue(Action.SMALL_ICON, icon);
    }
}

Action addAction = new MyAction("add");
JButton b = new JButton(addAction);
addAction.setEnabled(false);
Action – example

Action addAction =
    new AbstractAction("Add", new ImageIcon("add.gif")) {
        public void actionPerformed(AE e) {
            addItem();
        }
    };

JButton b = new JButton(addAction);
JMenuItem menuItem = commandMenu.add(addAction);
addAction.setEnabled(false);
Action

- The previous example is not good enough for your app;
- we also need:
  - mnemonic, tooltip description, …
  - I18N
  - …
Internationalization

- **Locale**
  - General Localization (numbers, sorting, etc.)

- **ResourceBundle**
  - Localization of Text
Locale

- Class Locale
  - Locale("language code", "country code")
  - Locale("sv","SE"), Locale("en", "US")

- Example
  - Locale.setDefault(new Locale("es","ES"));
Resource Bundle

- Useful for internationalization
- Collecting all strings in a "translation file"
- ResourceBundle class allows for lookup
- One translation file for each language
File: todo/ui/lang.properties:
ui.ok = Ok
ui.cancel = Cancel

File: todo/ui/lang_sv.properties:
ui.ok = Okej
ui.cancel = Avbryt

ResourceBundle rb =
    ResourceBundle.getBundle("player.ui.lang");
String okString = rb.getString("ui.ok");
JButton okButton = new JButton(okString);
java.util.ResourceBundle

- Be specific and clear! Like this:
  
  ui.menu.exit.name = Exit
  ui.menu.exit.mne = x

- Not like this:
  
  exit=Exit

- Document if necessary
Property files

- Naming convention for adaptation to Locale
  - basename_language_country_variant
  - basename_language_country
  - basename_language
  - Basename

- The most specific is used first!
Change during run?

- Why can it be a problem to change language during program execution?
Using a file path is not possible when running a program that's in a jar file.

The way to find images that are bundled in the jar file is to ask the Java class loader,
  - ClassLoader is the code that loaded your program
  - It knows where things are.
Icons

- At compile time, the icon (e.g. exit.gif) must be located next to <src>.java
- Icons also need to be in CVS
  - Location, next to source code

- JAR-safe Loading:

```java
ClassLoader cldr = this.getClass().getClassLoader();
java.net.URL imageURL = cldr.getResource("TODO/images/plus.gif");
ImageIcon addIcon = new ImageIcon(imageURL);
```