Assignment 2 - Ideas

- Actions
  - Buttons (Icons)
  - Menu

- Internationalization (I18N)
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
(Foo Bar)

- 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);
    }
}
// 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
  - …
…and Internationalization…
Internationalization

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

- ResouceBundle
  - 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
I18N

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?
ClassLoader

- 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 \(<\text{src}>.java\)
- 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);
  ```