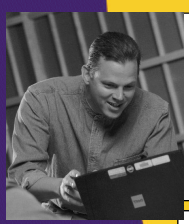
PrimeFaces: More Input Elements

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/jsf2/



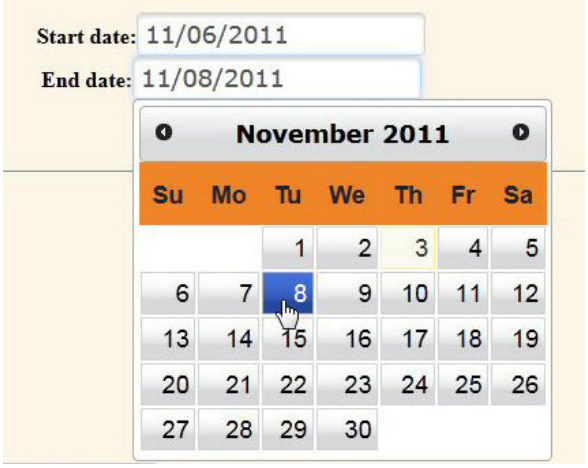
Topics in This Section

- **Date input**
  - p:calendar
- **Constrained String input**
  - p:input-mask
- **String input with suggestions**
  - p:autocomplete
- **Other input elements**
  - p:password, p:captcha, p:colorpicker, p:picklist, p:inputText

Date Input: p:calendar

p:calendar: Overview

- **Appearance and behavior**
  - Textfield with associated popup calendar.
- **Purpose: for collecting dates from user**
  - Value is automatically converted to Date. UI prevents user from entering an illegal value.



p:calendar: Summary of Most Common Attributes

- **<p:calendar …/>**
  - value
    - Should point to bean property of type Date.
  - mode (popup or inline)
    - Default is popup
  - showOn (focus, button, both)
    - When to display the calendar. Default is focus.
  - navigator (true or false)
    - Should menu be shown to navigate to month or year? Default is false.
  - pages
    - Number of months to show at once. Default is 1.

Example 1: Basics

- **Input page collects**
  - Start and end dates for resort checkin
    - Calendar pops up when user clicks in textfield
  - Example uses required and requiredMessage for validation. Also custom validation is performed in action controller, with message set if the end date is not later than the start date.
    - See earlier lecture on validation
- **Results page shows**
  - Confirmation of dates

Example 1: Bean (Bean Properties)

```
@ManagedBean
public class DateBean {
  private Date startDate, endDate;

  public Date getStartDate() {
    return(startDate);
  }

  public void setStartDate(Date startDate) {
    this.startDate = startDate;
  }

  public Date getEndDate() {
    return(endDate);
  }

  public void setEndDate(Date endDate) {
    this.endDate = endDate;
  }
```

Example 1: Bean (Formatted Date for Results Page)

```
public String getStartDay() {
  return(formatDate(startDate));
}

/** Given a Date, returns a String "Day, Month Number, Year",
 *  e.g. "Wednesday, November 2, 2011". For results page.
 */
private String formatDate(Date date) {
  if (date == null) {
    return("");
  } else {
    return(String.format("%tA, %tB %te, %tY",
                         date, date, date, date));
  }
}

public String getEndDay() {
  return(formatDate(endDate));
}
```


Example 1: Bean (Action Controller)

```
public String register() {
  FacesContext context = FacesContext.getCurrentInstance();
  if (!startDate.before(endDate)) {
    endDate = null;
    FacesMessage errorMessage =
      new FacesMessage("End date must be after start date");
    context.addMessage(null, errorMessage);
    return(null);
  } else {
    return("show-dates");
  }
}
```

Similar code for the end date.

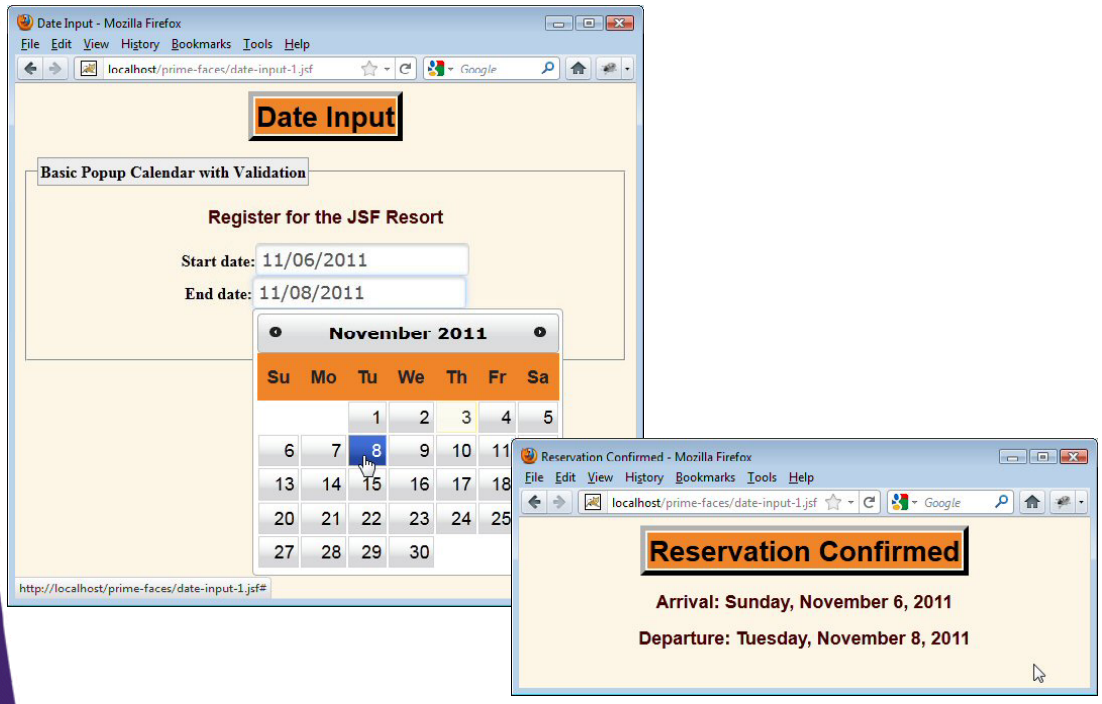
Example 1: Input Page

```
...
<h:form>
   <h2>Register for the JSF Resort</h2>
   <h:messages styleClass="error"/>
   <b>Start date:</b>
   <p:calendar value="#{dateBean.startDate}"
               required="true"
               requiredMessage="Start date required"/><br/>
   <b>End date:</b>
   <p:calendar value="#{dateBean.endDate}"
               required="true"
               requiredMessage="End date required"/><br/>
   <h:commandButton action="#{dateBean.register}"
                    value="Register"/><p/>
</h:form>
...
```

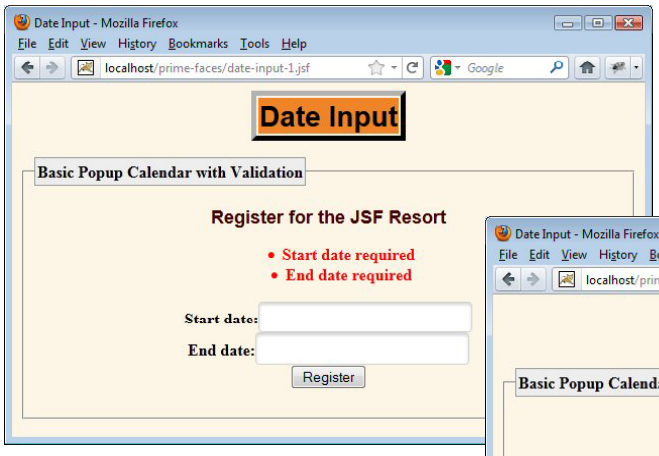

Example 1: Results Page

```
...
<h2>Arrival: #{dateBean.startDay}</h2>
<h2>Departure: #{dateBean.endDay}</h2>
...
```

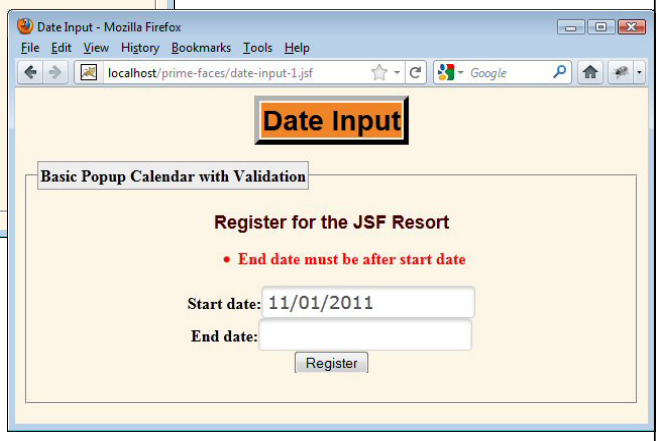
Example 1: Results (Good Dates)



Example 1: Results (Bad Dates)



Pressed button with both fields empty.



Pressed button with same date in both fields.

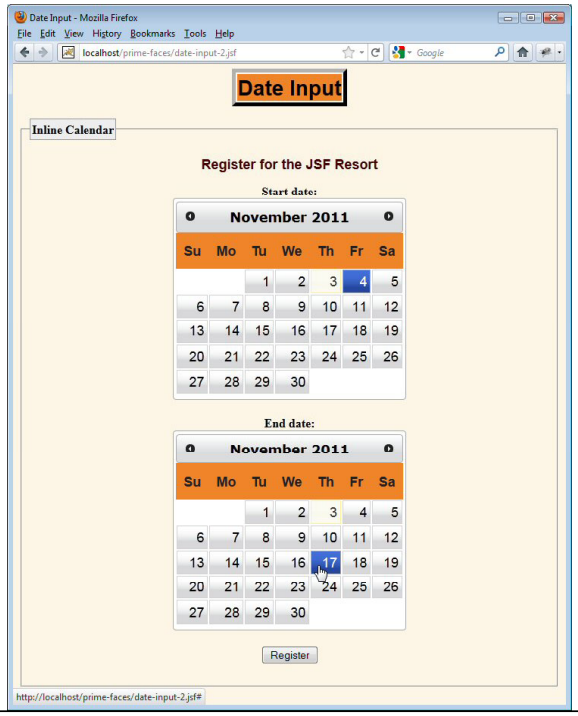
Example 2: Inline Calendar

- **Input page collects**
  - Start and end dates for resort checkin
    - Calendar is displayed inline, and no textfield is shown
- **Results page shows**
  - Confirmation of dates
- **Bean and results page**
  - Same as in previous example, so code not repeated here

Example 2: Input Page

```
...
<h:form>
   <h2>Register for the JSF Resort</h2>
   <h:messages styleClass="error"/>
   <b>Start date:</b>
   <p:calendar value="#{dateBean.startDate}"
               mode="inline"
               required="true"
               requiredMessage="Start date required"/><br/>
   <b>End date:</b>
   <p:calendar value="#{dateBean.endDate}"
               mode="inline"
               required="true"
               requiredMessage="End date required"/><br/>
   <h:commandButton action="#{dateBean.register}"
                    value="Register"/><p/>
</h:form>
...
```

Example 2: Results



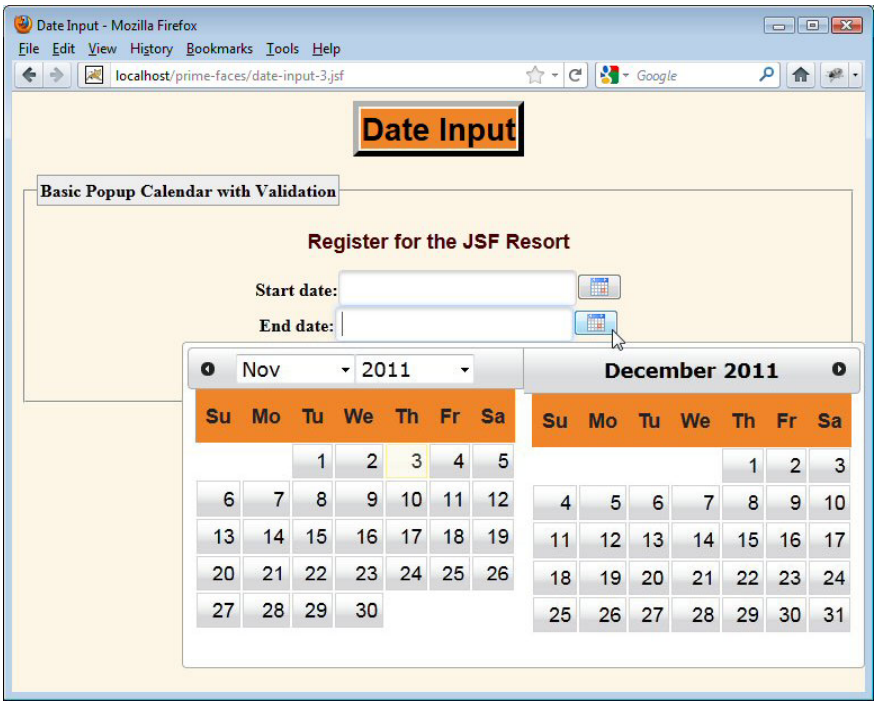
Example 3: Extra Button, Navigator, Multi-Month Display

- **Input page collects**
  - Start and end dates for resort checkin
    - Pops up either when user clicks in textfield or when small calendar button is pressed.
    - When calendar pops up, two months at a time are shown.
    - Menu lets you jump directly to specific month and year.
- **Results page shows**
  - Confirmation of dates
- **Bean and results page**
  - Same as in previous example, so code not repeated here

Example 3: Input Page

```
...
<h:form>
   <h2>Register for the JSF Resort</h2>
   <h:messages styleClass="error"/>
   <b>Start date:</b>
   <p:calendar value="#{dateBean.startDate}"
               showOn="both"
               navigator="true"
               pages="2"
               required="true"
               requiredMessage="Start date required"/><br/>
   <b>End date:</b>
   <p:calendar value="#{dateBean.endDate}"
               showOn="both"
               navigator="true"
               pages="2"
               required="true"
               requiredMessage="End date required"/><br/>
   <h:commandButton action="#{dateBean.register}"
                    value="Register"/><p/>
</h:form>...
```


Example 3: Results



Constrained String Input: p:input-mask

p:input-mask: Overview

- **Appearance and behavior**
  - Textfield with template text that constrains the values that the user can enter.
- **Purpose: for collecting strings from user**
  - Value is not converted. Bean property must be string. Spaces, dashes and other template text are sent with the string and must be parsed on server.
- **Documentation lacking**
  - PrimeFaces documentation is very poor. But uses the jQuery MaskedInput plugin, so read documentation there
    - http://digitalbush.com/projects/masked-input-plugin/

Phone: (___) ___-____

p:input-mask: Summary of Most Common Attributes

- **<p:input-mask …/>**
  - mask
    - Each character has four possible values
      - 9. Permits only a number to be entered there.
      - a. Permits only a letter (upper or lower case) to be entered there.
      - *. Permits a letter or a number to be entered there.
      - Anything else. Literal text that is displayed to the user and is not editable.
  - value
    - Should point to bean property of type String. Literal text in textfield (e.g., parens and dashes in phone number) is part of value sent to server, so server method must parse it

Example: Bean

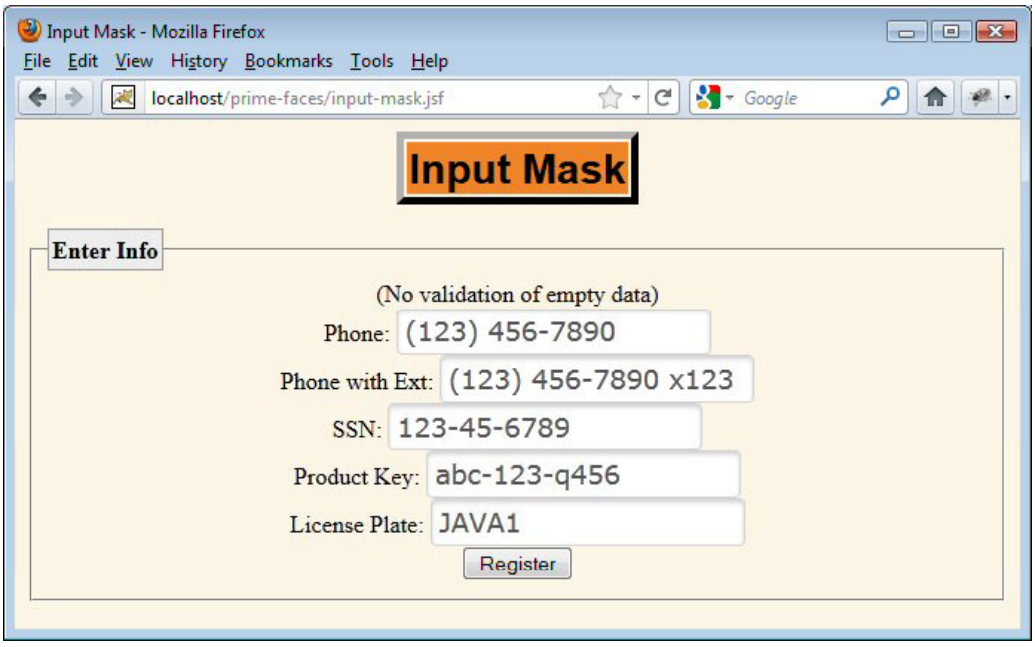
```
@ManagedBean
public class MaskBean {
  private String phone, phoneWithExt,
                 ssn, productKey, license;

  // Getters and setters for each
}
```

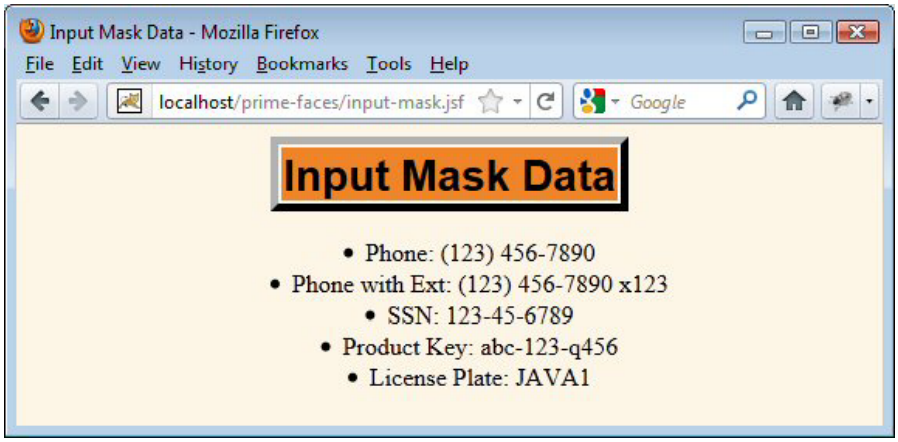
Example: Facelets

```
<h:form>
  Phone:
  <p:inputMask mask="(999) 999-9999"
               value="#{maskBean.phone}"/><br/>
  Phone with Ext:
  <p:inputMask mask="(999) 999-9999 x999"
               value="#{maskBean.phoneWithExt}"/><br/>
  SSN:
  <p:inputMask mask="999-99-9999"
               value="#{maskBean.ssn}"/><br/>
  Product Key:
  <p:inputMask mask="aaa-999-a999"
               value="#{maskBean.productKey}"/><br/>
  License Plate:
  <p:inputMask mask="*****"
               value="#{maskBean.license}"/><br/>
  <h:commandButton action="#{maskBean.register}"
                   value="Register"/>
</h:form>...
```


Example: Results (Form)



Example: Results (Results Page)



String Input with Suggestions: p:autocomplete

p:autocomplete: Overview

- **Appearance and behavior**
  - Textfield with dropdown of suggested completions
  - You can choose whether user is forced to accept one of the suggestions, or can also enter arbitrary text
- **Purpose: for collecting strings from user**
  - Value is not converted. Bean property must be string.

p:autocomplete: Summary of Most Common Attributes

- **<p:autocomplete …/>**
  - value
    - Should point to bean property of type String.
  - complete-method
    - A bean property referring to a server-side method that takes a String as input (the text entered so far) and returns a List<String> (the suggestions to show).
  - force-selection (true or false)
    - Is user constrained to choose a selection (true), or is free text allowed (false). Default is false.
  - minQueryLength
    - Number of chars before suggestions start. Default is 1.
  - queryDelay
    - Number of milliseconds before contacting server. Default 300.
  - Many more
    - p:autocomplete has many options, especially for Ajax.

Example 1: Constrained Input

- **Input page collects**
  - A computer programming language.
    - User can only choose a suggestion, and cannot enter languages not in the list (forceSelection="true")
    - Completions start after first character (default of 1 for minQueryLength)
    - Small delay after typing before server contacted (default of 300 for queryDelay)
- **Results page shows**
  - Confirmation of selection

Example 1: Bean (Suggestion Data and Bean Property)

```
@ManagedBean
public class LanguageBean {
  // 100 most popular programming languages, according to
  // http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
  // The first half are in order of popularity, the second half
  // are in alphabetical order.
  private static final String languageString =
    "Java,C,C++,PHP,C#,Python,...";
  private static final String[] languageArray =
    languageString.split(",");

  private String language;

  public String getLanguage() {
    return(language);
  }

  public void setLanguage(String language) {
    this.language = language;
  }
```

Example 1: Bean (Completion Method)

```
// Autocompleter method
public List<String> completeLanguage(String languagePrefix) {
  List<String> matches = new ArrayList<String>();
  for(String possibleLanguage: languageArray) {
    if(possibleLanguage.toUpperCase()
                       .startsWith(languagePrefix.toUpperCase())) {
      matches.add(possibleLanguage);
    }
  }
  return(matches);
}

// Action controller method

public String register() {
  return("show-language");
}
}
```

Unlike many client-side autocompleters, there is no builtin functionality as to how matches are done (front of text vs. middle of text, case sensitive, etc.). The method returns a List designating the suggestions, and the way the List is created is totally up to the developer. Here, I choose to do a case-insensitive match against the front of the user data.

Example 1: Facelets

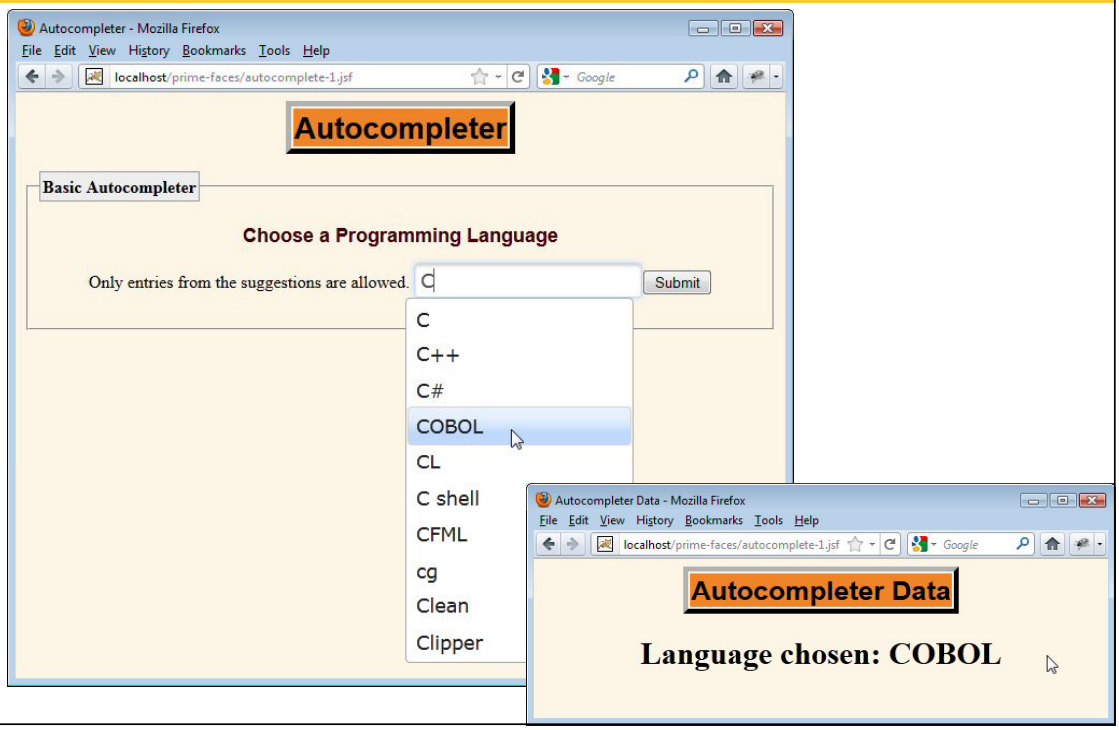
- **Input page**

```
<h:form>
   <h2>Choose a Programming Language</h2>
   Only entries from the suggestions are allowed.
   <h:messages styleClass="error"/>
   <p:autoComplete value="#{languageBean.language}"
                   completeMethod="#{languageBean.completeLanguage}"
                   forceSelection="true"
                   required="true"
                   requiredMessage="You must choose a language"/>
   <h:commandButton action="#{languageBean.register}"
                    value="Submit"/><p/>
</h:form>
```

- **Results page**

```
<h1>Language chosen: #{languageBean.language}</h1>
```

Example 1: Results



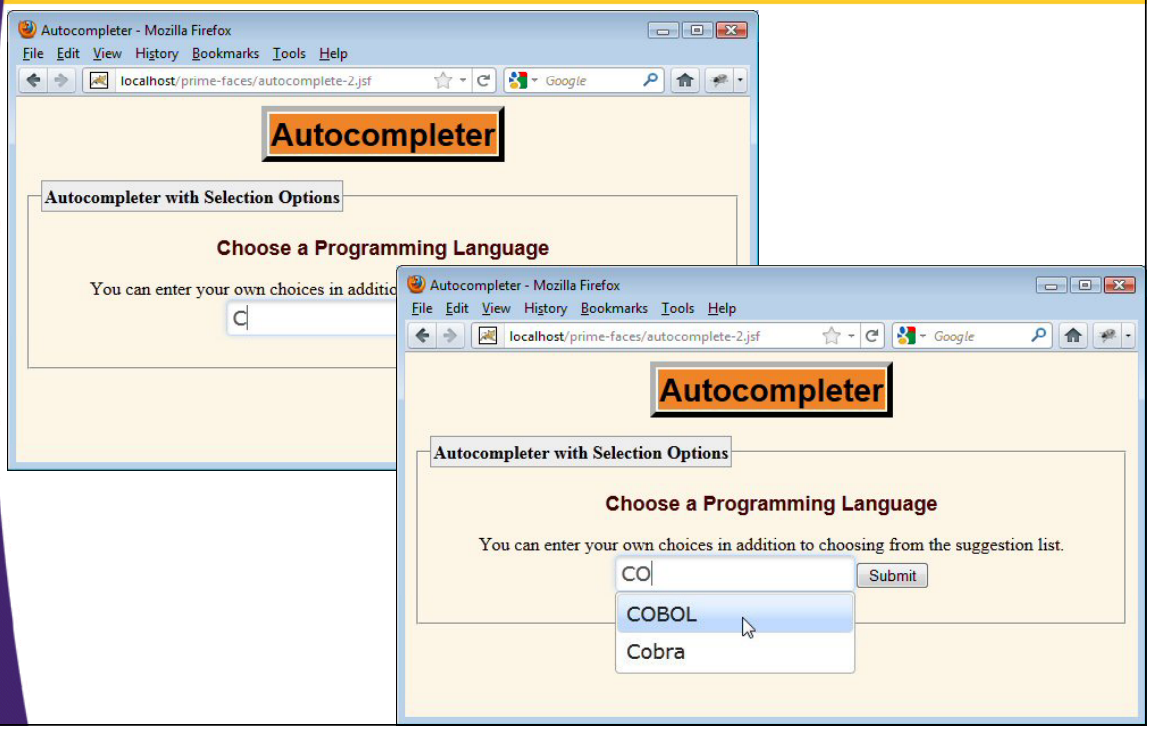
Example 2: Unconstrained Input

- **Input page collects**
  - A computer programming language.
    - User can either choose a suggestion or enter a language not in the list (default of false for forceSelection)
    - Completions start after second character (minQueryLength="2")
    - One second delay after typing before server contacted (queryDelay="1000")
- **Results page and bean**
  - Same as previous example, so code not repeated here

Example 2: Facelets (Input Page)

```
...
<h:form>
   <h2>Choose a Programming Language</h2>
   You can enter your own choices in addition to choosing
   from the suggestion list.
   <h:messages styleClass="error"/>
   <p:autoComplete value="#{languageBean.language}"
                   completeMethod="#{languageBean.completeLanguage}"
                   minQueryLength="2"
                   queryDelay="1000"
                   required="true"
                   requiredMessage="You must choose a language"/>
   <h:commandButton action="#{languageBean.register}"
                    value="Submit"/><p/>
</h:form>
...
```

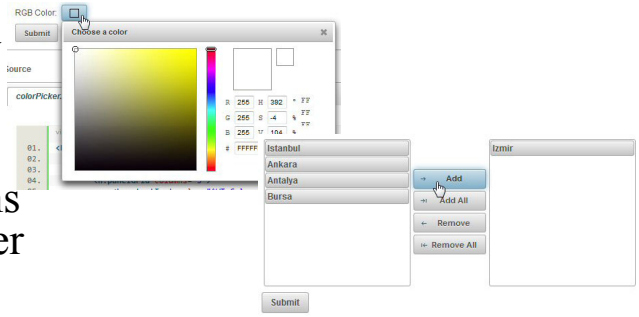

Example 2: Results



Other Input Components

Other PrimeFaces Components that Collect User Input

- **p:password**
  - Like h:inputSecret, but gives client-side feedback on password strength
- **p:captcha**
  - Lets user enter text to see if it matches graphics
- **p:colorpicker**
  - Lets user interactively choose a color
- **p:picklist**
  - Lets user move options from one list to another
- **p:inputText**
  - Adds theming (skinning) support to h:inputText



Wrap-Up

Summary

- **p:calendar**
  - <p:calendar value="#{someBean.date}" mode="…" showOn="…" pages="…" navigator="…"/>
- **p:input-mask**
  - <p:input-mask value="#{someBean.string}" mask="aa-99-**"/>
- **p:autocomplete**
  - <p:autocomplete value="#{someBean.string}" completeMethod="#{someBean.someMethod}" forceSelection="…"/>

Questions?