Object-oriented programming - lab in .NET environment

Lecture 04

Creating WindowsForms control

- In addition to the controls that come with the .NET framework, it is possible to create your own controls
- There are two types of controls we can create:
  - **User** controls
    - They are most often used
  - **Custom** controls

User controls

- User controls inherit from the class **UserControl**
- They consist of any **combinations of existing controls and components**
  - That is why they are also called **complex** or **composite** controls
- We add the user control to the form in the same way as other controls
- All controls in the user control are private
- We can add members to the user control:
  - Methods, properties, events, ...

Custom controls

- They inherit the class **Control**
- They provide the greatest possibility of adjustment
- The most demanding to develop
- Key feature: the control must draw itself
  - Drawing needs to be done by **overriding** method **OnPaint()**

Validation

- **Validation** is the procedure for checking the correctness of the data entered by the user in the form
- Component **ErrorProvider** is responsible for displaying a message about incorrect input
- Validation procedure:
  1. Let's set a property **CausesValidation** on **true** (default value) to the control that receives the user's input (eg TextBox)
  2. On the same control we implement a method to handle the event **Validating**
     1. If there is no error, we call the method **SetError()** with an empty string on the ErrorProvider
     2. If there are errors, we call the method **SetError()** with an error message on the ErrorProvider

Globalization and localization

- **Globalization**
  - Display of data (time, date, currency, number, ...) in formats adapted to a specific culture
  - For example:
    - In Croatia, we write the number as **1.800,00**
    - In the USA, the same number is written as **1,800.00**
- **Localization**
  - Displaying data in the language of a specific culture (translation)
  - For example:
    - The title of the form in Croatian can be "**Boja**"
    - A title of the same form in English can be "**Color**"

Culture

- Term **culture** in .NET implies the following elements:
  - Language
  - Alphabet (optional)
  - Region (optional)
- **Neutral culture**: contains only language information
  - Language: **hr, sr, en, ...**
- **Specific culture**: specifies language and region (can also be alphabet)
  - Language and region: **hr-HR, hr-BA, en-US, en-UK, en-CA, en-AU, ...**
  - Language, alphabet and region: **sr-Cyrl-BA, sr-Cyrl-CS, sr-Latn-BA, sr-Latn-CS, ...**

Culture change

- Culture is implemented in the class **CultureInfo**
  - It contains information about the format of time, date, currency, ...
- **Globalization** is set using:
  - Thread.CurrentThread.**CurrentCulture**
  - By default, the culture of the selected regional settings is applied
- **Localization** is set using:
  - Thread.CurrentThread.**CurrentUICulture**
  - By default, the culture (language) of the operating system is applied
  - Specifies which resources will be loaded into localized forms

Localized forms

- Each form can have multiple versions - one for each desired culture
  - We choose culture using **CurrentUICulture**
  - Once selected, the application will load the resources of the selected culture
  - If the resources do not exist, it will load the resources of the default culture
  - Culture selection must be done before displaying the form
    - It is also possible to dynamically change the localized strings on the form
- Essential properties of the form for localization:
  - **Localizable** – if set to value **true**, the designer stores the properties of forms and controls in resource files
  - **Language** – the designer displays the selected localized version of the form

Printing

- The component responsible for printing is **PrintDocument**
- It works as as follows:

1. On the instance of the class **PrintDocument** we define the method for handling events called **PrintPage**
2. On the instance of the class **PrintDocument** method **Print()** is called
3. The **PrintPage** event will be raised for the first page
   - Printing on a paper is done using **Graphics** class
   - At the end of the printout, the value of the property **HasMorePages** is set on an instance of the class **PrintPageEventArgs**
4. As long as **HasMorePages** equals **true**, the **PrintPage** event is raised
5. When the document is sent for printing, the **EndPrint** event is raised

Additional controls for printing

- When printing, we can use additional controls:
  - **PrintDialog**: represents a dialog for selecting the printer, the pages you want to print, the number of copies, ...
  - **PageSetupDialog**: presents a dialog for selecting paper size, page orientation, margins, ...
  - **PrintPreviewDialog**: represents a dialog for previewing the document before printing
  - **PrintPreviewControl**: used to develop your own dialog for previewing the document before printing
- All the listed controls are connected to **PrintDocument** component via **Document** property