Quick recap

A reminder of what we talked about during Lecture 02

Last time…

- Desktop applications
  - What are they?
- Web applications
  - What are they?
  - How they differ from desktop apps?
  - What is the common architecture of web apps?
  - What technology can be used?
- Development
  - What should be taken into consideration?
  - What are agile methods and lean?

Web application development with Vaadin

Lecture 03

Getting started with Vaadin: Eclipse, Tomcat and `Hello, world!`

Overview

- System requirements
- Eclipse setup
- Hello, world
- Understanding the code
- Debugging
- Adding interaction
- Deploying the project
- Summary

System requirements

What do we need

Technology

FIXED

- Java SDK
  - Java EE 1.6+
  - Development
- Java servlet container
  - Deployment
- Compatible web browser
  - Testing
  - Using

FLEXIBLE

- Operating system
  - Windows
  - Linux
  - MacOS X
- Servlet container
  - Any will do
- Web browser
  - Any with Java Script will do

SUGGESTED

- Java EE IDE
- Debug tool for web pages
- Source code repository

Why Eclipse

- Vaadin Plugin
  - Official support
- Build system support
  - IvyDE
  - Required by Vaadin Plugin
- Code formatting
  - Official Vaadin rules

Eclipse setup

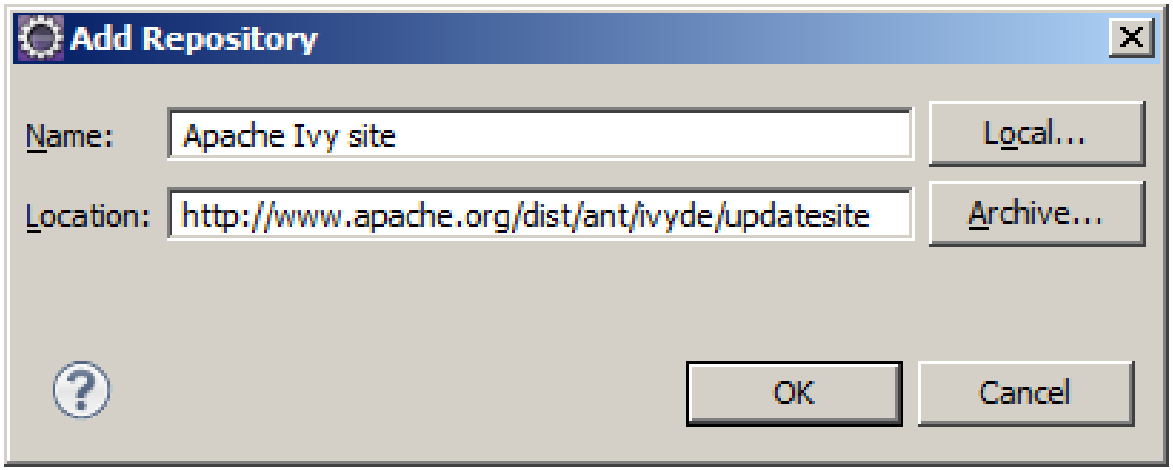
Downloading, installing and configuring

1. Downloading Eclipse

- Eclipse for Java EE Developers
  - 200+ MB
  - **Not** Eclipse for Java Developers
    - ~150 MB
  - Check OS version before downloading
- Unzip somewhere easily accessible
- Run
  - Specify workspace location

2.1 Installing IvyDE

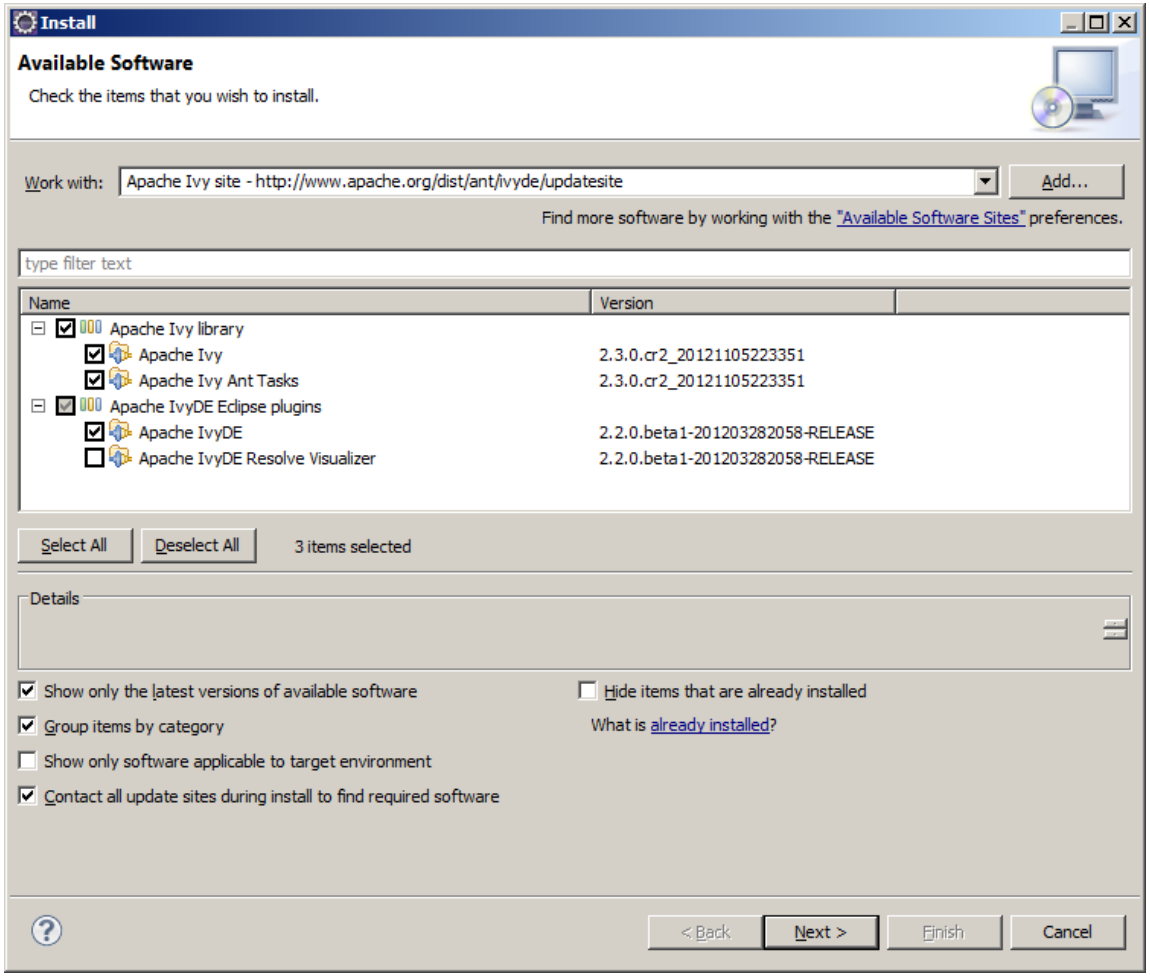
- Help → Install new software… → Add…
  - Name (e.g.): Apache IvyDE update site
  - Location: http://www.apache.org/dist/ant/ivyde/updatesite



  - OK

2.2 Installing IvyDE

- Select
  - Apache Ivy library
    - Apache Ivy
    - Apache Ivy Ant Tasks
  - Apache IvyDE
    - Apache IvyDE
- Next, Next
- I accept…
- Finish, OK, Yes



3. Installing Vaadin plugin

- Help → Install new software.. → Add…
  - Name (e.g.): Vaadin Experimental Plugin site
  - Location: http://vaadin.com/eclipse/experimental
  - OK
- Select Vaadin → Vaadin plug-in for Eclipse
- Next, Next, I accept…, Finish, Yes

4.1 Setting up Eclipse: Code formatting

- **Required** in this course
  - And when you contribute code to Vaadin community
- File encoding and delimiters
  - Window → Preferences → General → Workspace
    - Text file encoding → Other → UTF-8
    - New text file line delimiter → Other → Unix
- Formatting and cleanup
  - Visit http://dev.vaadin.com/svn/versions/6.8/eclipse/
  - Save both xml files somewhere
  - Window → Preferences → Java → Code style
    - → Clean up → Import → `VaadinCleanup.xml`
    - → Formatter → Import → `VaadinJavaConventions.xml`
- Java compiler
  - Window → Preferences → Java → Compiler → Errors/Warnings
    - Potential programming problems → Serializable class without… → Ignore

4.2 Setting up Eclipse: Save actions

- Window → Preferences → Java → Editor → Save actions → Perform selected actions
  - Format source code
  - Organise imports
  - Additional actions → Configure
    - Code organising
      - Select *Remove trailing whitespace for all lines*, deselect else
    - Code style
      - Select *Use blocks in… always*, deselect else
    - Missing code
      - Select everything except *Implementations of interface methods…*
    - Unnecessary code
      - Select *Remove unused imports*, *Remove unneccesary casts*, deselect else

5.2 Configuring Eclipse web browser

- Window → Preferences → General → Web browser
  - Use external web browser
  - Override the default one if you want

5.1 Installing Tomcat

- Help → Install new software… → Work with → Eclipse WTP repository
  - Select the latest version of WTP SDK
  - Next, Next, I accept…, Finish, Yes
- Window → Preferences → Server → Runtime Environments → Add…
  - Apache → Apache Tomcat v7.0
  - Next
  - Download and Install
    - I accept…, Finish
  - Select installaction directory
  - OK

6. Other things to configure

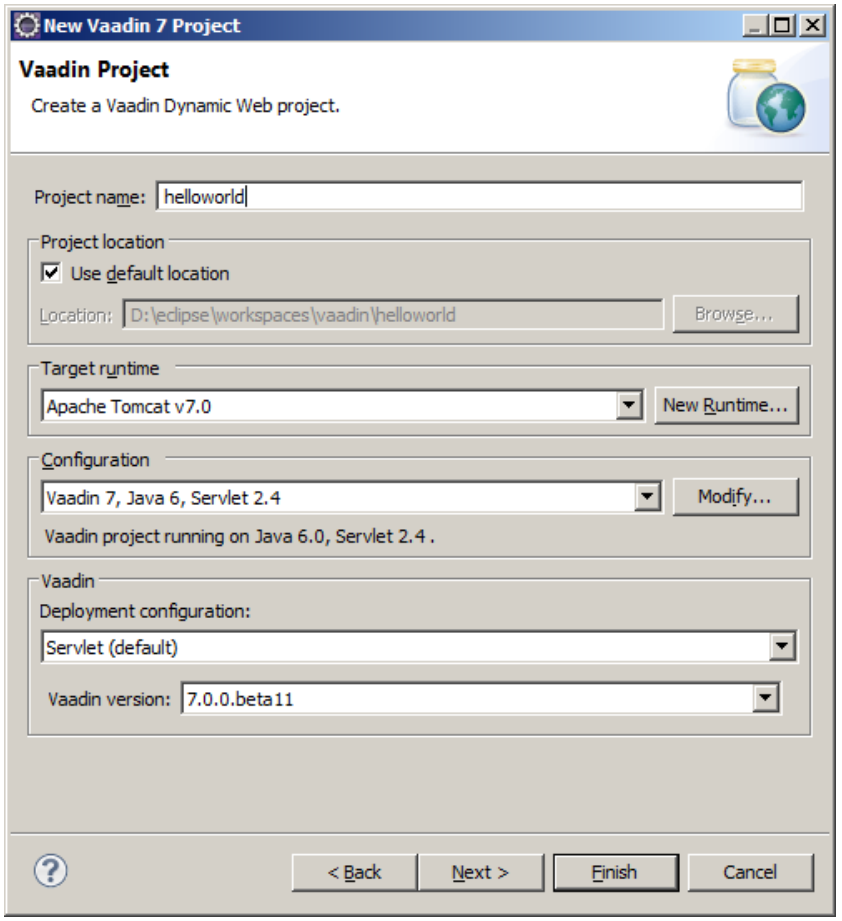
- Code repository tools
  - Subversive for SVN
  - Egit for git
  - Eclipse Trac Plugin to connect with Trac
- Syntax highlighting
  - White background?
- Line numbers
  - Window → Preferences → General → Text editors → Show line numbers

Hello, world!

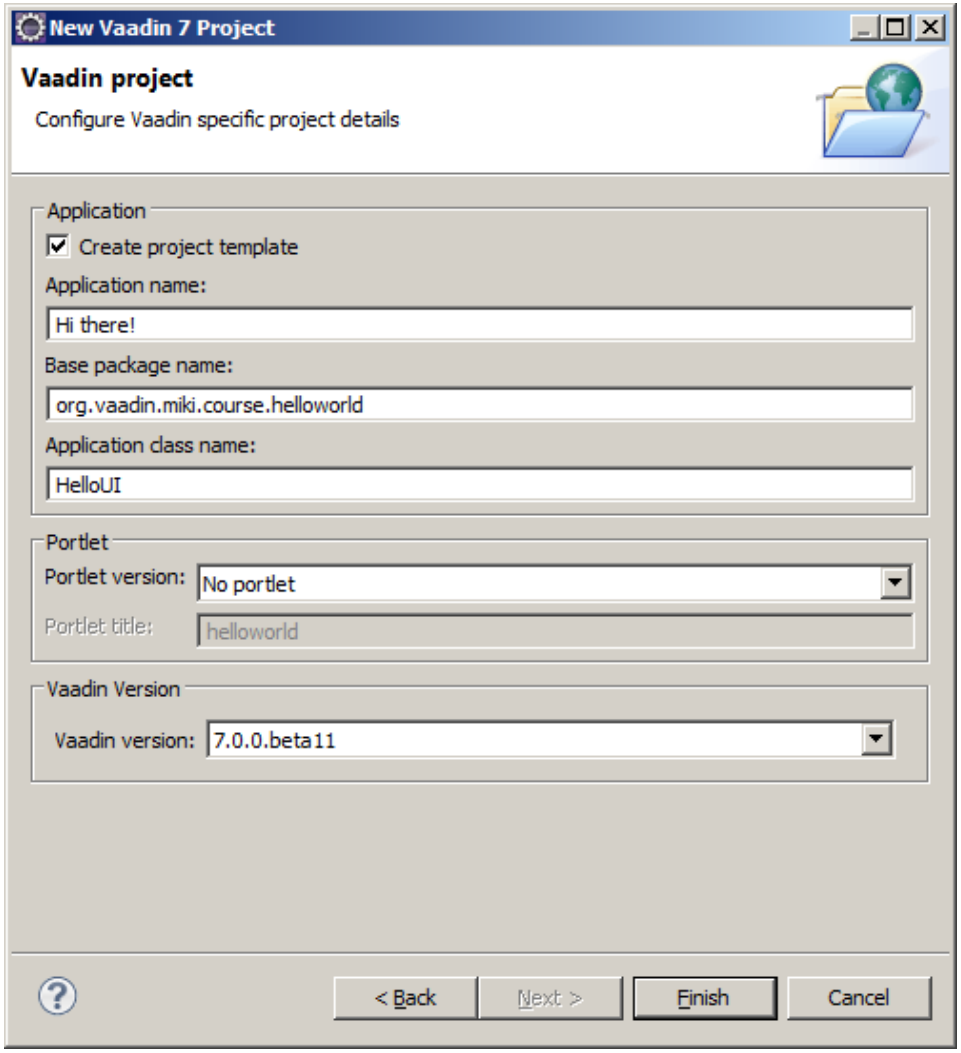
Or rather: Click Me!

1.1 New project: basic setup

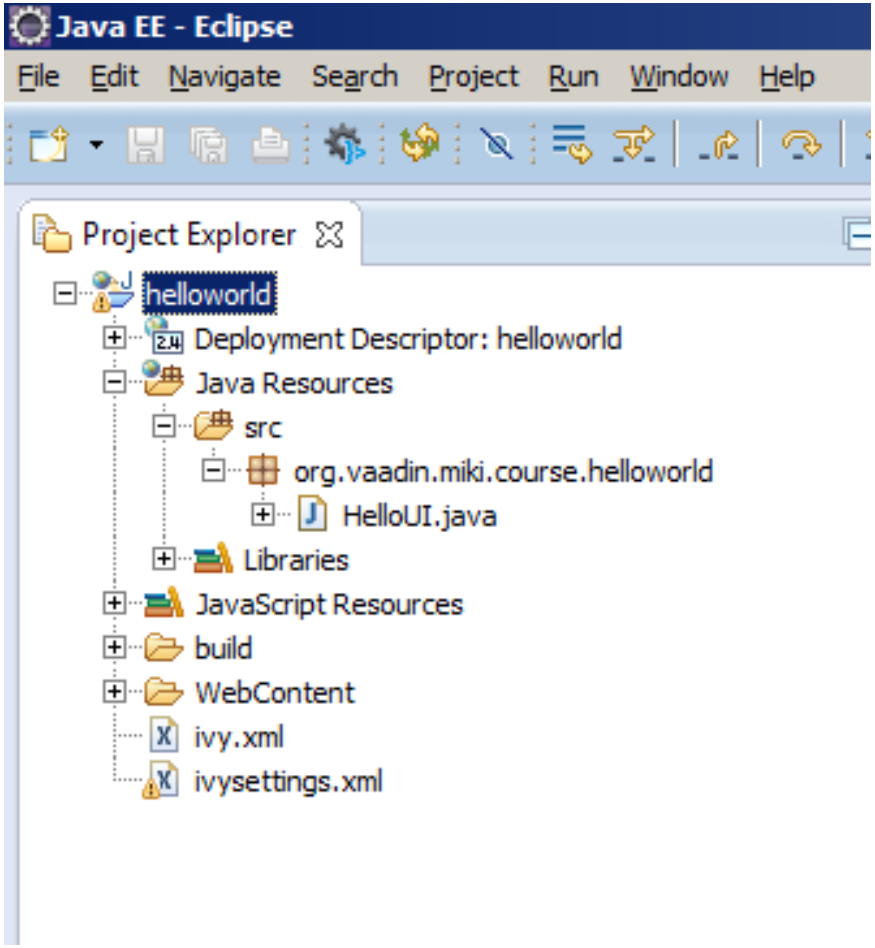
- File → New → Other… → Vaadin → Vaadin 7 Project
  - Type in project name
  - Target runtime → Apache v7.0
  - Configuration → Vaadin 7, Java 6, Servlet 2.4
  - Vaadin version: **latest**
    - Dev team promised .rc1
    - If not, use 7.0.0.beta11
- Next, Next, Next



1.2 New project: app template



2. Project structure



3. App source code

```
1.  package org.vaadin.miki.course.helloworld;
2.
3.  import com.vaadin.server.VaadinRequest;
4.  import com.vaadin.ui.Button;
5.  import com.vaadin.ui.Button.ClickEvent;
6.  import com.vaadin.ui.Label;
7.  import com.vaadin.ui.UI;
8.  import com.vaadin.ui.VerticalLayout;
9.
10. @SuppressWarnings("serial")
11. public class HelloUI extends UI {
12.
13.       @Override
14.       protected void init(VaadinRequest request) {
15.                   final VerticalLayout layout = new VerticalLayout();
16.                   layout.setMargin(true);
17.                   setContent(layout);
18.                   Button button = new Button("Click Me");
19.                   button.addClickListener(new Button.ClickListener() {
20.                               public void buttonClick(ClickEvent event) {
21.                                           layout.addComponent(new Label("Thank you for clicking"));
22.                               }
23.                   });
24.                   layout.addComponent(button);
25.       }
26. }
```

4. Starting up and closing

- Right-click project name in project explorer
  - Run → Run on server
  - Select Tomcat v7.0
  - Select *Always use this server…*
  - Next, Finish
- Click the button
- Terminate server
- Click the button again
  - Observe the error
- Close browser window

Understanding the code

Line by line (almost)

The source code (again)

```
1.  package org.vaadin.miki.course.helloworld;
2.
3.  import com.vaadin.server.VaadinRequest;
4.  import com.vaadin.ui.Button;
5.  import com.vaadin.ui.Button.ClickEvent;
6.  import com.vaadin.ui.Label;
7.  import com.vaadin.ui.UI;
8.  import com.vaadin.ui.VerticalLayout;
9.
10. @SuppressWarnings("serial")
11. public class HelloUI extends UI {
12.
13.         @Override
14.         protected void init(VaadinRequest request) {
15.                     final VerticalLayout layout = new VerticalLayout();
16.                     layout.setMargin(true);
17.                     setContent(layout);
18.                     Button button = new Button("Click Me");
19.                     button.addClickListener(new Button.ClickListener() {
20.                                 public void buttonClick(ClickEvent event) {
21.                                             layout.addComponent(new Label("Thank you for clicking"));
22.                                 }
23.                     });
24.                     layout.addComponent(button);
25.         }
26. }
```

The source code explained

- `public class HelloUI extends UI {`
  - UI is a base class for any application
- `protected void init(VaadinRequest request) {`
  - This is where the execution *starts*
- `final VerticalLayout layout = new VerticalLayout();`
  - Layouts arrange components
    - Lecture 04 is about components
    - Lecture 06 is about layouts
- `setContent(layout);`
  - Sets the contents of the application main view
- `Button button = new Button("Click Me");`
  - Components are created like regular Java objects
- `button.addClickListener(new Button.ClickListener() {`
  - Components usually broadcast events that can be captured
- `layout.addComponent(button);`
  - Adding components to layouts is trivial

Characteristics

- (Almost) like a desktop Java app
  - Events
  - Listeners
  - Layouts
- Nothing webapp-specific
  - So far

Debugging

Briefly about how to trace the execution

1. Place breakpoints

- Navigate to a line in the source code
  - Be it creating the label when button is clicked
- Toggle breakpoint
  - Ctrl+Shift+B
    - Pro-tip: buy a keyboard with assignable macro keys ☺
  - Or right-click margin and toggle

2. Debug

- Right-click project name
  - Debug as → Debug on server
  - Server-side code debugging
  - Click the button
- Controlling the execution
  - Step into = F5
  - Step over = F6
  - Step return = F7
  - Continue = F8
- Close browser and terminate server

Adding interaction

Magically send the data entered by the user

1. Create text field

```
final TextField field =
    new TextField(
      "Enter your name");
layout.addComponent(field);
```

2. Modify click listener

```
layout.addComponent(
    new Label(
     "Thank you for clicking, " +
     field.getValue()
  ));
```

3. Test

- Save changes
- Right-click project name
  - Run as → Run on server
- Enter anything in the text field
- Click the button
- Clear the text field
- Click the button

4. Improve click listener

```
String name = field.getValue();
if(name.isEmpty()) name =
    "anonymous user";
layout.addComponent(
    new Label(
      "Thank you for clicking, " +
       name));
```

5. Test

- Save changes
- Wait for Eclipse to reload Tomcat
- Reload web page
- Click the button

The code

```
15.  public class HelloUI extends UI {
16.
17.  @Override
18.  protected void init(VaadinRequest request) {
19.    final VerticalLayout layout = new VerticalLayout();
20.    layout.setMargin(true);
21.    setContent(layout);
22.
23.    final TextField field = new TextField("Enter your name");
24.    layout.addComponent(field);
25.
26.    Button button = new Button("Click Me");
27.    button.addClickListener(new Button.ClickListener() {
28.      public void buttonClick(ClickEvent event) {
29.        String name = field.getValue();
30.        if(name.isEmpty()) name = "anonymous user";
31.        layout.addComponent(new Label("Thank you for clicking, "+name));
32.      }
33.    });
34.    layout.addComponent(button);
35.  }
36.
37.  } //class
```

Deploying the project

Moving project to an external servlet container

As simple as it can be

- Right-click project name
- Export → WAR File
  - Specify location
- Deploy the WAR to a container

Summary

What did we do today?

Lessons of today (hopefully)

- Dependencies
  - What is required to develop Vaadin apps?
  - What is optional?
- Setting up Eclipse
  - How to install Vaadin plugin?
- Writing Vaadin applications
  - What is the main class?
  - How to capture events with listeners?
  - How to debug server-side code?
  - How to deploy the application?

Coming up next

- Overview of components
- Basic principles of UI/UX design
- Layouts, themes and styles

Suggestions? Questions?

miki@vaadin.com
t: @mikiolsz