Today

- About last year’s evaluation
- Administration
  - Registration
  - Groups (3 or 4 student/group)
  - CVS
- Diary
- Homepage
  - [http://www.it.uu.se/edu/course/homepage/devgui2/vt09](http://www.it.uu.se/edu/course/homepage/devgui2/vt09)
- Assignment 1
About last year’s course

- Positive
  - Overall ok (4.35)
  - Assignments (4.18)
  - The course style/setup

- Lacking
  - Lots of work (3.76, mean was 84h)
  - Clearer assignments
  - Grading per assignment?
Examination

- No written exam
- A lot of the work in groups
- Group meetings are mandatory
- Grading on:
  - The group's overall result on the assignments
  - Individual diary
  - Individual activity and ability to show what you have learned during the group meetings
  - Presentation of assignments
Diary

- Work must be reported this way, including all time spent on the course
- Part of each assignment
  - Submitted in the JAR file
Aim and goals

- Similar setup as in UID 1
  - Assignments (4?)
  - Group work
  - Group meetings (6)
  - Presentations (NEW)

- Purpose
  - Doing things outside of standard Swing
  - Design Patterns
  - Programming the ”right way” and critical thinking
  - Future GUIs (NEW)
Important

- Form groups
- Get your CVS working
  - You will get a new player to start with

- Send email to erik.boralv@it.uu.se

- One mail per group:
  - Group members: first name, last name
  - CVS username
  - CVS pass
ASSIGNMENT 1
Media Button Factory

- `myButton = new MediaButton(MediaButton.PLAY, playAction);`

- **A reusable button**
  - No bitmap from file - instead, generate a bitmap at runtime
  - Many states (play, stop, next)
  - Must handle changes to foreground and background color (at runtime) AND resize

- **Use Factory (for the reusable images)**
Multi-functional button
Threads

- **Definition**: A thread is a single sequential flow of control within a program.

**Typical program: single-threaded**

**Multi-threaded program (typical Java Swing program)**

A Thread

Two Threads
Swing is single-threaded!

- Once a Swing component has been realized, all code that might affect or depend on the state of that component should be *executed* by the event-dispatching thread.
Implications when using Swing

- The most important thing is to make sure that the Swing thread won't hang when time-consuming operations are performed.
- It is also important that calls to Swing always are made from Swing's event-thread.
Are we in the Event-thread?

- If unsure, you can always test:
  - if (EventQueue.isDispatchThread())

- But... unnecessary to check:
  - public void actionPerformed(ActionEvent e)

- We are always in the event-thread when we receive an event from Swing.
Why is Swing single-threaded?

- Complicated to make a toolkit thread-safe.
  - Multithreaded toolkits: A failed dream?
  - [http://weblogs.java.net/blog/kgh/archive/2004/10/multithreaded_t.html](http://weblogs.java.net/blog/kgh/archive/2004/10/multithreaded_t.html)

- Not many developers are able to handle multi-threaded user interfaces.

- Events are received in a predictable sequence (non-deterministic behavior avoided)

- Faster UI (no need to synchronize)
Won´t work

public class Delay extends JFrame {
    private JLabel label = new JLabel("Go on - push the button!");

    public Delay() {
        getContentPane().setLayout(new BorderLayout());

        JButton beep = new JButton("Beep");
        beep.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("Working...");
                try { Thread.sleep(3000); } catch (Exception e1) {} label.setText("Done!");
            }
        });

        getContentPane().add(beep, BorderLayout.NORTH);
        getContentPane().add(label, BorderLayout.SOUTH);
        pack(); show();
    }
}
Solution

```java
public class Delay2 extends JFrame {
    private JLabel label = new JLabel("Go on - push the button!");

    public Delay2() {
        getContentPane().setLayout(new BorderLayout);

        final JButton beep = new JButton("Beep");
        beep.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("Working...");
                beep.setEnabled(false);
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        try { Thread.sleep(3000); } catch (Exception e1) {}  
                       label.setText("Done!");
                        beep.setEnabled(true);
                    }
                });
            }
        });
        getContentPane().add(beep, BorderLayout.NORTH);
        getContentPane().add(label, BorderLayout.SOUTH);
        pack();show();
    }
}
```
Threading tips

- Timer and Thread – avoid writing your own solutions. You will fail!
- Check out `javax.swing.SwingWorker` for more advanced options
- This will do just fine:
  ```java
  Runnable doWorkRunnable = new Runnable()
  { public void run() { doWork(); }
  }
  SwingUtilities.invokeLater(doWorkRunnable);
  ```
- `java.util.Timer vs. javax.swing.Timer`
final Runnable doUpdateCursor = new Runnable() {
    boolean shouldDraw = false;
    public void run() {
        if (shouldDraw = !shouldDraw) {
            drawCursor();
        } else {
            eraseCursor();
        }
    }
};

Runnable doBlinkCursor = new Runnable() {
    public void run() {
        while (isCursorBlinking()) {
            try {
                EventQueue.invokeLater(doUpdateCursor);
                Thread.sleep(300);
            } catch (InterruptedException e) {
                return;
            }
        }
    }
};

new Thread(doBlinkCursor).start();
Swing Timer

Action updateCursorAction = new AbstractAction() {
    boolean shouldDraw = false;
    public void actionPerformed(ActionEvent e) {
        if (shouldDraw = !shouldDraw) {
            drawCursor();
        } else {
            eraseCursor();
        }
    }
};

new Timer(300, updateCursorAction).start();