Graphic Interface Programming II

Events and Threads
Animation

- Animation adds to user experience
- Done right, it enhances the User Interface
- Done wrong, it distracts and irritates
Active Objects

- `javax.swing.Timer`

- `Timer t = new Timer(interval, listener);`
  - Interval is in milliseconds
  - Listener is an object listening to the events
    - `ActionListener`

- Don’t confuse with `java.util.Timer` (!)
Concurrency
Synchronization

- Making sure that some events can’t collide
- Preventing deadlocks
- Preventing data corruption

EXAMPLES
Threads in Swing

- Swing is NOT thread-safe!

- Using concurrency works most of the time, but...
  ...might cause unpredictable errors...
  ...that are difficult to reproduce....

- Event dispatch thread
Threads

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

- 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.
Event Dispatch Thread

Event Dispatch Thread (EDT)

Event Queue

Event

Event

...

Event

java.awt.EventQueue

Event dispatch

Requestor

Requestor
EDT

- A series of "small" tasks in a queue
  - e.g., invocations of event-handling methods
  - scheduled from application code (invokeLater)
  - javax.swing.Timer

- Ensures that the tasks are performed in a "safe" manner
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.
Safety rules

- Drawn Swing-components should only be changed by Listener methods
- Never call directly on a Listener method
- Components can be changed **before** they are drawn
  - but not after ”pack()” and ”setVisible(true)"
- If we need to access Swing components after drawing
  - `invokeLater();` and `invokeAndWait();`
  - These two methods schedule into the EDT.
Are we in the Event-thread?

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

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

- We are always in the event-thread when we receive an event from Swing.
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);
```
Example: Consumer

```java
public class Consumer implements Runnable {
    public Thread activity = new Thread(this);
    private long interval;
    private Queue q;

    public Consumer(long time; Queue k) {
        interval = time * 1000;
        q = k;
    }

    public void run() {
        while (true) {
            try {
                Thread.sleep(interval);
            } catch (InterruptedException e) { break; } // interrupt the loop
            System.out.println("Some text");
        }
    }
}
```
Example: Consumer 2

public Consumer(long time, Queue q, JTextArea a) {
    interval = time * 1000;
    q = q;
    a = a;
}

public void run() {
    Runner r = new Runner(); // create dispatched class
    while (true) { try { Thread.sleep(interval); } // wait for interval.
        catch (InterruptedException e) { break; } // interrupt the loop
        SwingUtilities.invokeLater(r); // dispatch to EDT
    }
}

private class Runner implements Runnable { // internal class
    public void run() {
        // interface method
        a.append(q.getFirst() + " "); // accessing JTextArea!
    }
}
Why is Swing single-threaded?

- Complicated to make a toolkit thread-safe.
  - [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)
Swing Graphics

- Everything is painted from scratch
  - `paintComponent(Graphics g);`

- Don’t call `paintComponent(g);` directly!
  - `(allowed as super.paintComponent(g); )`

- Call to `repaint();` puts request into EDT!
Painting and paintComponent

```java
public void paint(Graphics g) {
    paintComponent();
    paintBorder();
    paintChildren();
}
```
repaint();

- repaint(); is called to force a redraw of the component

- Requests a redraw to be added to the event

- The redraw is scheduled to the paint manager
Double Buffering

- Principle for updating screen graphics
- Smoother animation
- Enables more complex graphics
- Handled automatically by Swing
  - You need not be concerned about it
  - You should know the principles
Double Buffering

Single Buffering

Double Buffering
Course Assignment

- The course "book"!

- The assignment is designed to
  - provide you with challenges
  - give you new knowledge
  - illustrate techniques
  - be "real" to some extent
  - (hopefully) be fun and inspiring
Learning from the course

You can lead a camel to the water.
But you can’t force it to drink!

- You are responsible for your own learning!
- I am responsible for showing you “the water”
- If you get stuck…
  - ask someone
  - help each other
Learning from the course

= 

Programming!!!
Learning from the course

= 

Programming!!!
Lots of programming!!!!!
Learning from the course

= 

Programming!!!
Lots of programming!!!!!
Lots and lots and lots of programming!!!!!!!
Programming

- If you want to know how something works?
  • Test it!

- If you want to see if you can do ”like this”?
  • Test it!

- If you have a great idea?
  • Test it!
Course assignment

- Your "test track"
- Your playground
- A stage for setting up your "Java play"
- A reason to do some programming
Course assignment

- Simple overall task:

- Extend the TODO-application to a full-featured Calendar-application

- The assignments will make this extension in progressive steps
Basic Layout Inspiration
Assignment 1

- Will be carried out over Easter (holiday)

- Starting from the TODO-application
  - (reuse as much as possible)

- Create a Calendar Layout
  - Using Standard Layout Manager for the Monthly view
  - Not a Table!!!
Days

- Each Day is a "Day Card" (Jpanel)

- Each Day Card should:
  - contain the necessary information about the day
  - be resizable
  - have a custom graphic design using `paintComponent( )`; and decorations
    - (a design made by the group)
Java Interface

- The Day card class needs to implement an interface

- The interface should be designed by you
  - You have to decide what functionality is needed
  - Motivate your choice!

- Part of the assignment
An Animation Driver Class

- Start on the **design** of a class that will be used to run animations
  - Don’t implement yet!
  - Pilot (experiment) implementations are allowed
- Threads or Timers?
- How can a single class (instances) drive several animations simultaneously?
- How should an Animation Driver Class communicate with the Animation?
Overall Concerns

- Coupling and Cohesion!
- Documentation!
- Design motivations!
- (Documentation?)
- Flexibility in solutions
- Keep Programming!
- (Documentation?)
- KISS (Keep it simple, Sam!)
- Make it OOP!
Happy Easter(Egg)Hacking!
Happy Easter(Egg)Hacking! or Easter Egg Hatching?