Design Patterns

+ Animation
  Undo/Redo
  Graphics and Hints
Design Patterns

• Design:
  – the planning
  – that lays the basis for the making
  – of every object
  – or system

• Pattern:
  – a type of theme
  – of recurring
  – events or objects
Design Patterns in OOP

• Recipes for solving standardized problems

• Abstractions of concrete problems
  – *How to make a class that allows only one instance of itself:*
    • The Singleton Design Pattern
    • Provides a general solution that needs to be implemented
Started from Architecture

• Architectural Patterns
  – Lecture rooms
  – Studies
  – Town Squares
  – Restaurants
  – Etc.

• Adopted in many non-related areas
Design patterns

• describe...
• how objects communicate...
• without becoming entangled...
• in each other’s data models...
• and/or methods.
What is a design pattern?

• A general recipe

• A framework

• A template

• Pre-made solution
Design Pattern Content

• Intent – what is the goal of the pattern?
• Applicability – when should it be used?
• Structure – the pattern description
• Consequences – pros and cons of using the pattern.
Example

• The ”Singleton pattern”
• Ensures that one and only one instance can be created from a class

```java
private static Singleton m_instance;
private Singleton() { ... } // private constructor

public static synchronized Singleton getInstance()
{
    if (m_instance == null) // only create new first time
        m_instance = new Singleton();

    return m_instance; // return the single instance
}
```
Design Patterns vs. Libraries

- Libraries contain concrete classes
  - Implement concrete solutions

- Design Patterns are abstract solutions
  - Advice on general solutions

- You can’t import a design pattern
DP Factory

• Another interesting Design Pattern

• What is a factory?
  – Production unit
    • produces instances
    • multiple instances of different classes
  – Ask for a type and you get it!
Factory Example

• A graphic Shape
  – circles, rectangles, stars, etc.
  – All shapes have size, and can be drawn!

• The Shape interface defines methods

• Use a ShapeFactory:
  
  public Shape getShape(String type, int size) {...}
ShapeFactory, Why?

• One class provides all shapes!

• Dynamic creation!

• Consistent usage of Shapes
  – Unless explicitly needed, we don’t have to know which Shape we have created afterwards
UNDO

• The Undo/Redo mechanism should provide:
  – Single Undo
  – Single Redo (of Undone activity)

• and, if possible:
  – Undo and redo of several consecutive action

• How can this be achieved?
Command Design Pattern

- One way to handle queues of actions!
- Important DP for the operation of UNDO
- Simple principle
A very simple interface!
  – One method
  
  execute()
Command

• All activities are driven by the `execute()` method
  – Encapsulate all user activities in ”Command objects”
Simple example: Macro

```java
public class IAm implements Command {
    public void execute() {
        System.out.println("I'm the command pattern!");
    }
}

// An object that holds commands:
public class Macro {
    private List commands = new ArrayList();

    public void add(Command c) {
        commands.add(c);
    }

    public void run() {
        Iterator it = commands.iterator();
        while (it.hasNext()) {
            ((Command) it.next()).execute(); // Casting!
        }
    }
}
```
UNDO?

• The `execute()` method describes what happens during an edit activity...
• UNDO means reversing the editing activity...
• In the command interface add:
  – `unexecute()`
  – `reexecute()`
  – Describes the backward process

```java
public interface Command {
    public void execute();
    public void unexecute();  // undo
    public void reexecute();  // redo
}
```
UndoManager

• UndoManager is support for UNDO in Swing

• "edit types" - the effect of a user invoked command
  – each edit type must still have a definition of the edit and its effects

• Simplifyes the management of the Undo/Redo
Swing Undo

• UndoableEdit (Interface)
• AbstractUndoableEdit (Abstract class)
• CompoundEdit (class for sequences of undoables)
• UndoableEditListener (Interface)
• UndoableEditEvent (notification object)
• UndoableManager (Queue manager)
  – Like EventManager
• UndoableEditSupport (Support class)
Undo/Redo-queue

UndoManager

op1  op2  op3  op4
Undo/Redo-queue

UndoManager

op1  op2  op3  op4

op1  op2  op3  op4
Undo/Redo-queue

UndoManager

op1  op2  op3  op4

op1  op2  op3  op4

op1  op2  op5

op1  op2  op5
Undo

- Good example program in linked page on course homepage (Assignment 2)

Add an undo/redo function to your Java apps with Swing
Animation Driver Class

• Many Good Suggestions (these will be commented groupwise in separate email)

• Good example of a Design Pattern

• How to ensure OOP principles?
Purpose of Animation Driver

• One or more Animations
  – Simulation
  – Realtime game (e.g. Battlebots)
  – Other dynamic realtime animation with several animated objects

• One Actor controls the Animation process
  – Singleton (suggested by one group)
  – Controls Animation Progress
Decisions?

• The amount of control in driver?

• Different animation speeds?

• Which objects to animate?

• How is the driver controlled?
OOP Principle

• The driver needs to be free of knowledge about the respective animations

• Each animated object responsible for its own animation, e.g.:
  – Driver controls general animation speed
  – Animated object controls its own relative animation speed
Animator Pattern

• Interface AnimatedObject
  – One primary method: tick()
  – Each tick is one cycle in general Animation

• Animator
  – Tracks (manages) the animated objects
  – Sends ticks to all its Animated Objects
  – Methods:
    • addObject(AnimatedObject ao)
    • removeObject(AnimatedObject ao)
    • tick()   // sends ticks to all animated objects
Animator Pattern

• Animated Object could be any class
  – As long as it implements the AnimatedObject interface
• Does not have to be a graphic component!
  – In fact it is easier to draw if it is not...
• A panel can host many animated graphics
  – defines only paintComponent(Graphics g)!
  – the graphic adds its own drawing to the Graphics environment!
Animated Graphics

• Uses the paintComponent of (e.g.) Jpanel

• Implements Drawable interface:
  – method: draw(Graphics g)
  – the method should specify the graphic commands for the graphic

• Does **not** specify the paintComponent
  – is not a Component, but a ”loose” graphic
Jpanel paintComponent

• Each Drawable object implements the draw() method.

```java
@Override
public void paintComponent(Graphics g) {
  Drawable d;
  for (Iterator<?> it = graphicObjects.iterator(); it.hasNext(); )
    d = (Drawable) it.next();  // Casting to Drawable
    d.draw(g);
}
```
Complex graphics

- Every graphic uses `draw(Graphics g)` to add itself to the graphics environment
- A single call to `paintComponent()`
Animated Complex Graphics

• If we want to implement graphics that move, the class has to implement both interfaces.

```java
public class MovingStar implements Drawable, AnimatedObject {
  ...
  // here the drawing is made in draw(Graphics g)
}
```
Animation Process

• Create graphic Container (JPanel?)
• Collect all graphic objects (not Swing Components)
• For every tick:
  – update the graphic objects (position, size, colour, etc.)
  – In paintComponent(Graphics g) draw all graphic objects on the Graphics environment g
(Detour)

Mixing graphics and Swing components?
Don’ts in GUIs

• We will be able to do almost anything in GUI design

• But some things maybe we shouldn’t

• (but no rule without exception, of course)
Move things for the user

- You **can** control the mouse pointer for the user
  - java.awt.Robot library
    - java.awt.Robot
    - mouseMove(int x, int y);

- Should you?
  - Most of the time – NO!
  - Bad GUI!

- One possible exception: Guiding help systems
The Robot is still Useful!

Can make screenCaptures!

createScreenCapture(Rectangle screenRect)

returns a BufferedImage
Remove Title Bar of Window

• You can remove the Titlebar of a window

• Should you?
  – Maybe, e.g. in AboutBoxes

• But remember to provide the user with control of the window!
Use Animations

• You can use animations

• Should you?
  – Yes, sometimes
  – But restrictively

• Beware of perception exhaustion
Use Undo

• You will be able to Undo things

• Should you?
  – Yes, but only significant changes
  – Don’t undo single character insertion

• What is significant?
Use graphic effects

• Swing allows you to use graphic effects
  – E.g. GradientPaint, Transparency

• Should you?

• Yes, but always provide an alternative skin that is clear and distinct
  – Make it easy to access

• Make sure to test that the effects work on the platform
Be Creative

• It is (in principle) your imagination (and available time) that sets the limits for what you can do.

• Should you?
  – Yes, as long as it is good User Interface Design
  – Yes, as long as it is possible to understand
  – Yes, as long as it makes sense for the intended user groups
Practice Animation

• You have a large toolbox with graphics! Practice using it!

• Should you?

• Yes, even if it not all will be used in the Calendar, the more you practice, the more you will know about how and when you should use it!