Basics of programming 3

Java GUI and SWING

Complex widgets

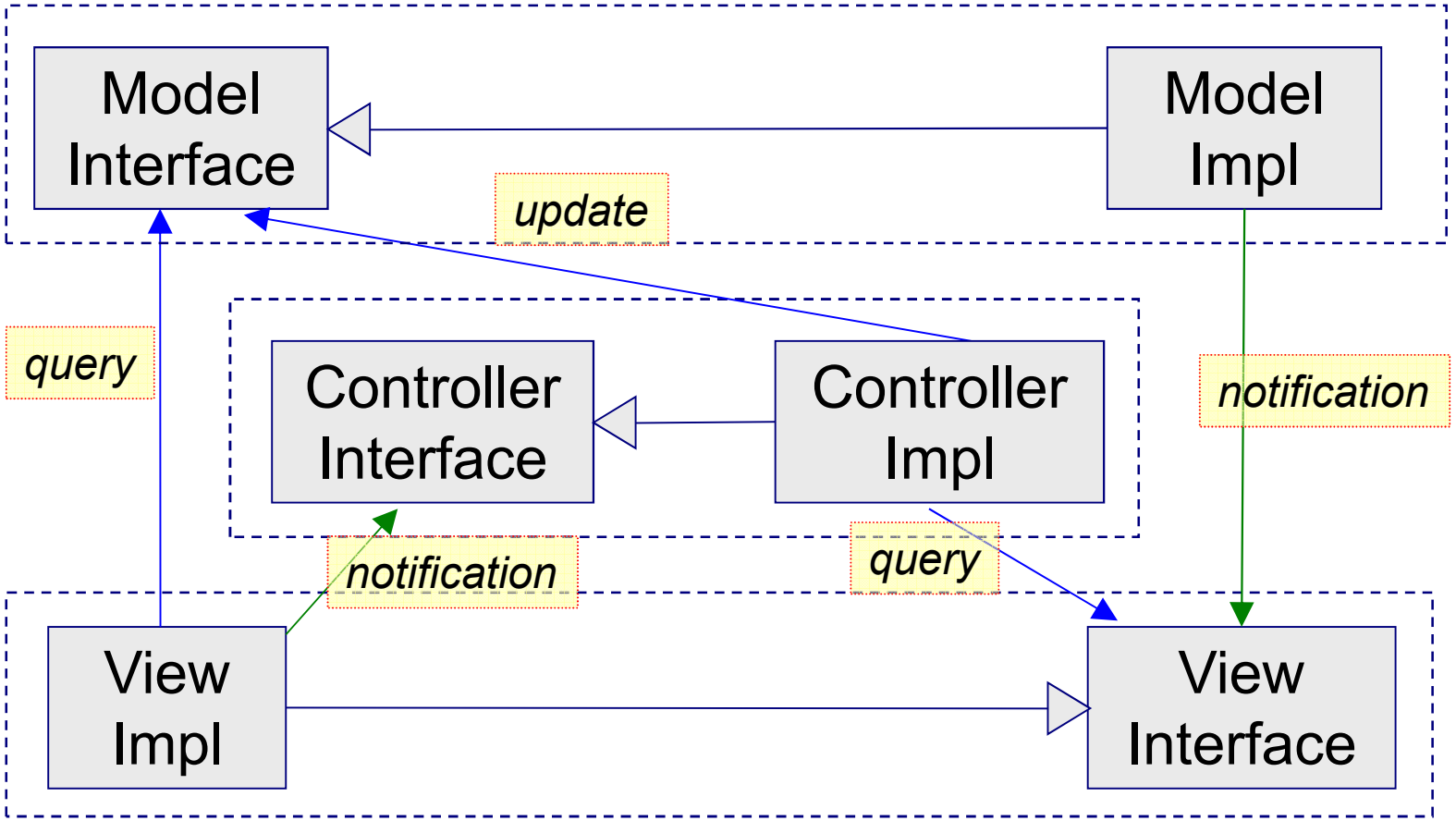
Complex widgets

- JList
  - elements can be selected from a list
- JComboBox
  - drop down list with optional textfield
- JTable
  - matrix-like representation of data
  - more on laboratory
- JTree
  - tree-like representation of hierarchical data
- …

Separation of aspects

- Model-View-Controller (MVC) pattern
  - model: *storing and handling data*
  - view: *GUI representation and visual info*
  - controller: *event handling by listener implementation*
- Simple widgets
  - combine data and GUI, only listeners are separated
- Complex widgets
  - GUI and data are also separated
    - e.g. *JList* and *ListModel*, *JTable* and *TableModel*
    - default model implementations are provided

MVC architecture



Complex widget: `JList`

- Shows a list of objects
  - by default objects' *toString()* method is used
- Objects are stored separately
  - *Object[]* (immutable)
  - *Vector<T>* (immutable)
  - *ListModel* implementation (eg. *DefaultListModel*)
- Scrolling is done by *JScrollPane*
  - *JScrollPane* is a container providing scrolling functionality
  - *JList* only renders the list

Model part: `ListModel`

- **`interface javax.swing.ListModel`**
  - **`Object getElementAt(int index)`**
    - returns element at *index*
  - **`int getSize()`**
    - returns number of stored elements
  - **`void removeListDataListener (ListDataListener l)`**
  - **`void addListDataListener (ListDataListener l)`**
    - adds/removes listener

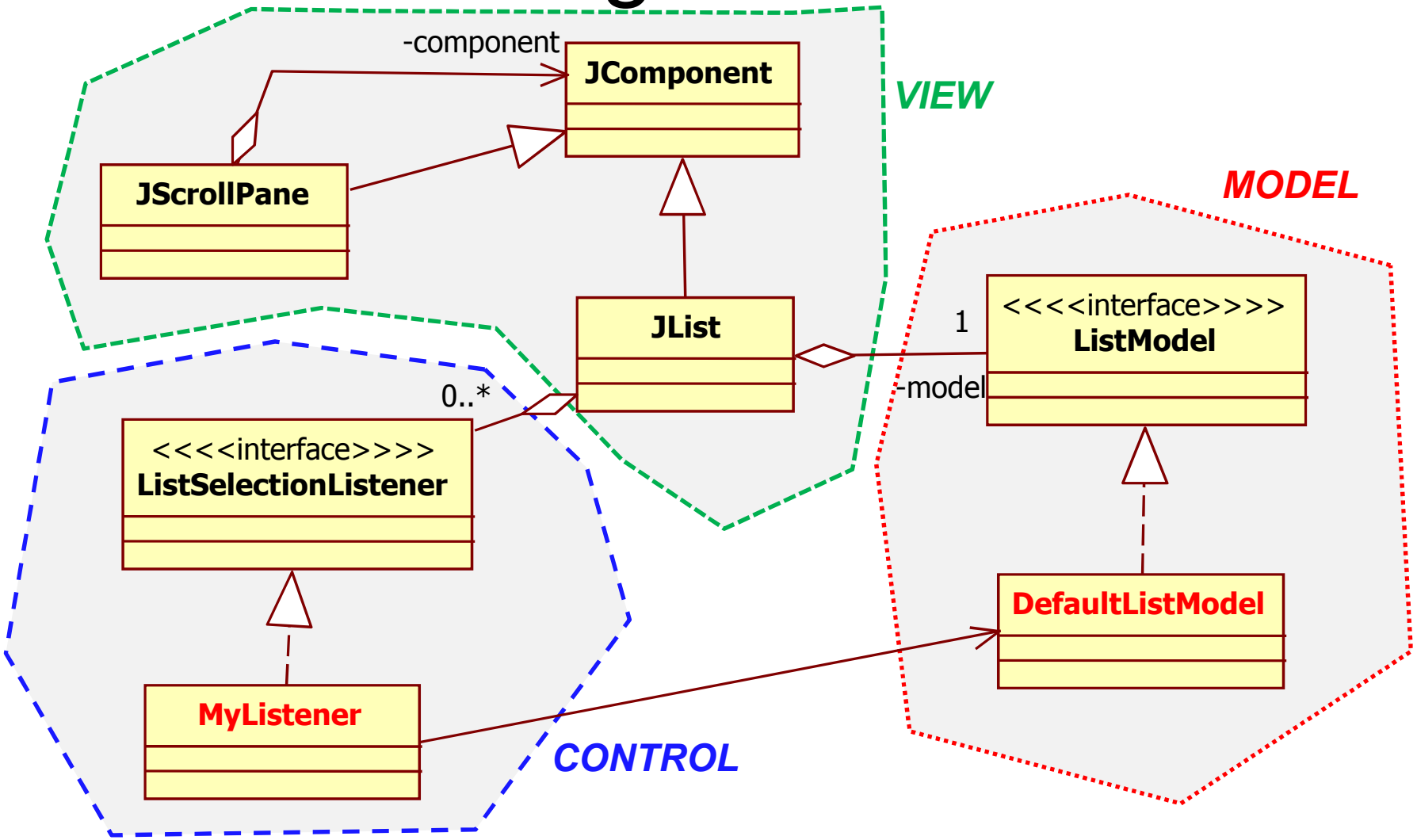
`ListDataListener`

- When model changes, listeners are notified
- *JList* has its own implementation:
  - **`BasicListUI.ListDataHandler`**
- Methods
  - **`void intervalAdded(ListDataEvent e)`**
  - **`void intervalRemoved(ListDataEvent e)`**
    - interval specified in *e* is modified
  - **`void contentsChanged(ListDataEvent e)`**
    - complex change has occured

DefaultListModel

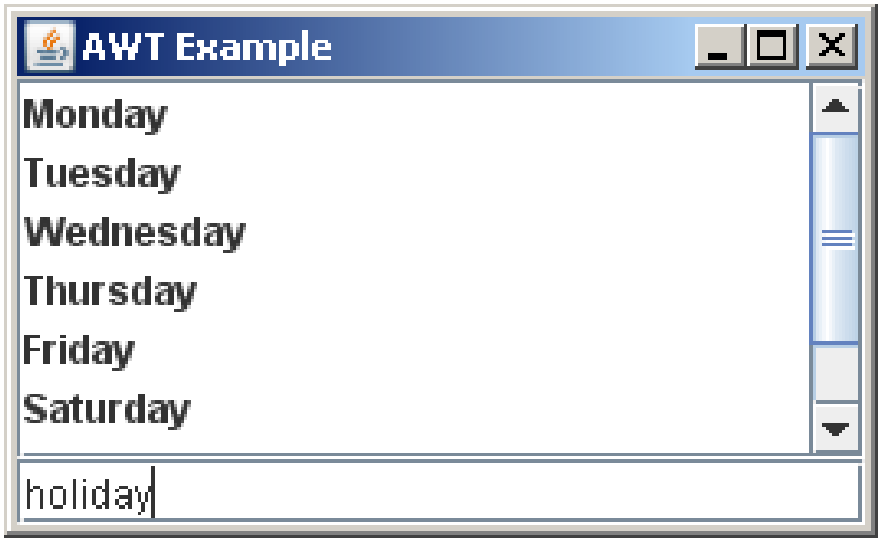
- Implements interface *ListModel*
- Methods resembling *java.util.List*
  - `void add(int index, Object o)`
  - `int size()`
  - `Object get(int index)`
  - `Object remove(int index)`
  - ...

JList class diagram



JList example

- Let's create a window with
  - a *list* for displaying strings
  - a *textfield* for entering strings
    - when entered, list is updated



JList example code

```
public class SwingList implements  ActionListener{
    JTextField tf;
    JList list;
    DefaultListModel model;
    public void actionPerformed(ActionEvent ae) {
        model.addElement(tf.getText());
        tf.setText("");
    }
    static public void main(String args[]) {
        (new SwingList()).run();
    }
    ...
```

JList example code

```
    ...
    public void run() {
        JFrame f = new JFrame("AWT Example");
        tf = new JTextField("", 20);
        model = new DefaultListModel();
        list  = new JList(model);
        JScrollPane pane = new JScrollPane(list);
        tf.addActionListener(this);
        f.add(tf, BorderLayout.SOUTH);
        f.add(pane, BorderLayout.CENTER);
        f.pack();
        f.show();
    }
}
```

MVC in operation

- If model is modified, GUI is updated
  - calling model.addElement(“Holiday”)
  - makes GUI refresh
- If GUI gets an event, Controller is notified
  - *ListSelectionListener*
    - `void valueChanged(ListSelectionEvent e)`
  - For retreiving selected value(s), use
    - *JList’s* `getSelectedIndex()` or `getSelectedIndices()` and
    - *ListModel’s* `getElementAt()` methods

MVC analysis

- **Model**: **`DefaultListModel`**
  - handles collection of elements
  - when changes, notifies *View*
- **View**: **`JList`***`(+BasicListUI.ListDataHandler)`*
  - displays elements only
- **Controller**: **`SwingList`** *`(implements ActionListener)`*
  - updates *Model* based on GUI element (*View*) actions

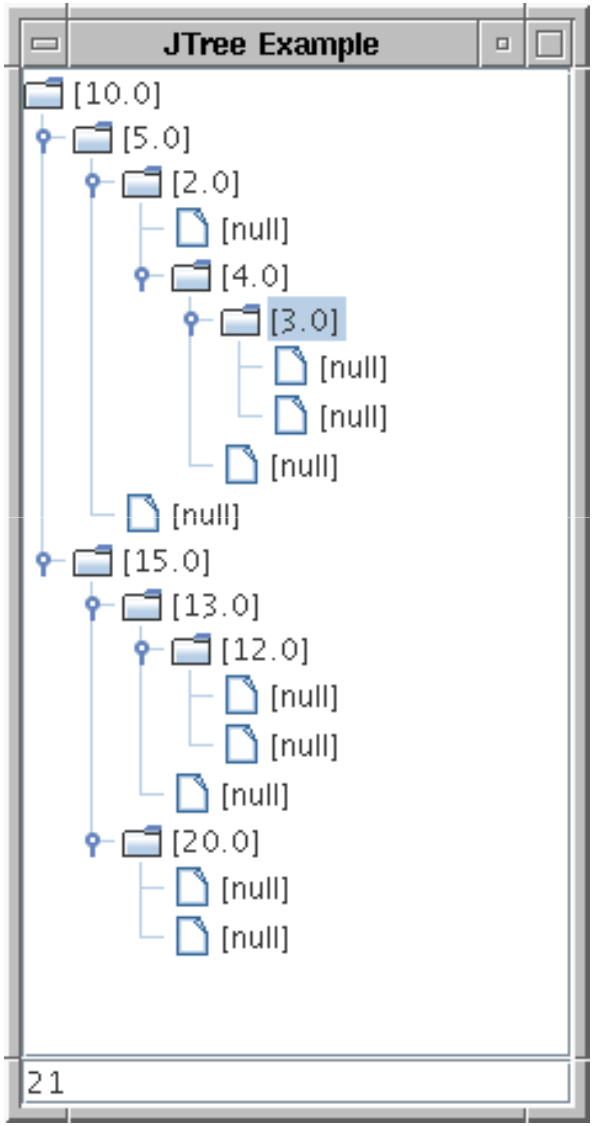
Complex widget: **JTree**

- Displaying tree structure (hierarchy)
- Model: **`TreeModel`**

- Might be initialized with a **`TreeNode`**-tree
  - when specifying the root

JTree example

- **Problem**: let's create an application for displaying a binary tree!
  - tree stores doubles
  - new element can be added



JTree example

- **`BinTree`**
  - implements a binary tree (DIY)
  - no children: uses *Null* object instead of *null*
    - easier implementation
    - *Null Object pattern*
    - similar to sentinels in linked lists
    - expensive!
  - after insertion full path to new element is to be returned

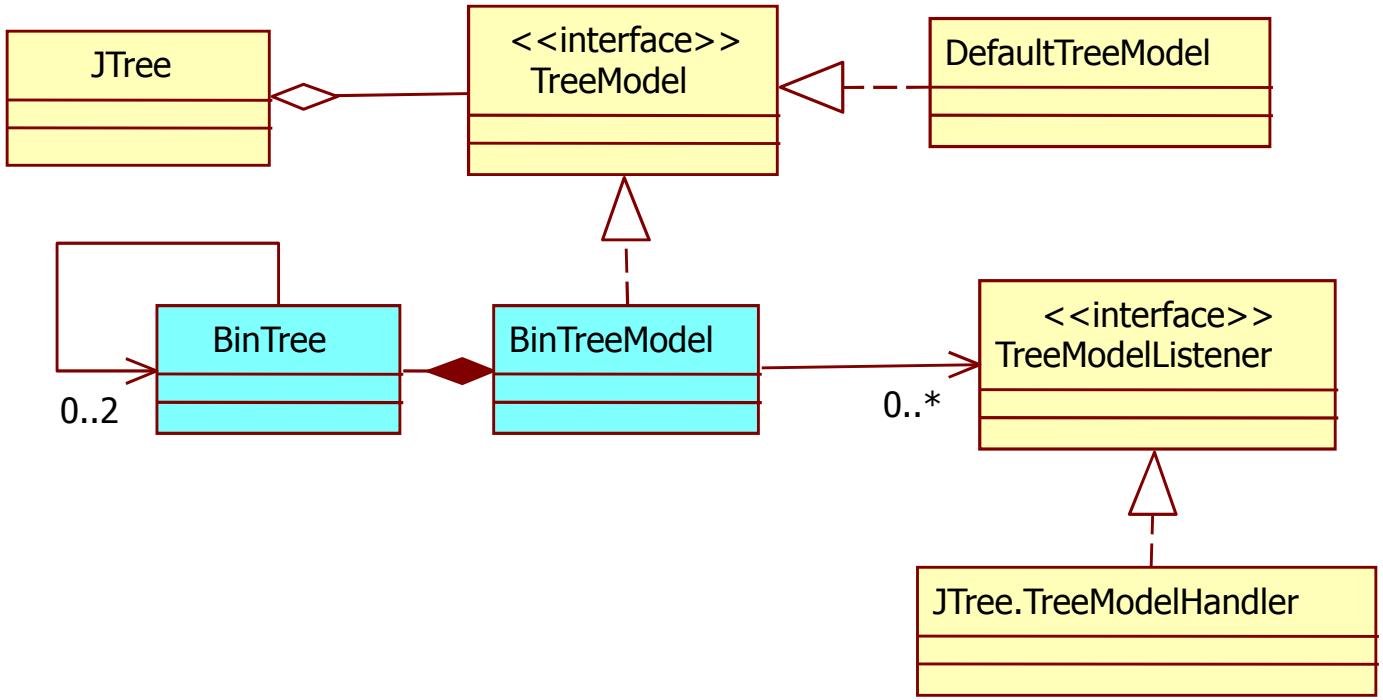
JTree example

- **`BinTreeModel`**
  - the model for **`JTree`**
  - gives access to **`BinTree`**
    - **`BinTree root = new BinTree()`**
  - handling **`TreeModelListener`**-s
    - **`Vector<TreeModelListener> listeners`**
    - **`public void addTreeModelListener(TreeModelListener l)`**
    - **`public void removeTreeModelListener(TreeModelListener l)`**

JTree example

- **`BinTreeModel`** (cont)
  - accessing the model
    - `public Object getChild(Object parent, int index)`
    - `public int getChildCount(Object parent)`
    - `public int getIndexOfChild(Object parent, Object child)`
    - `public Object getRoot()`
    - `public boolean isLeaf(Object node)`
    - *`public void valueForPathChanged(TreePath path, Object newValue)`*
    - `public void insert(double d)`

JTree class diagram



JTree example: application

```
JTextField tf;
BinTreeModel btm;
```

```
public void actionPerformed(ActionEvent ae) {
  double d = Double.parseDouble(tf.getText());
  btm.insert(d);
  tf.setText("");
}
```

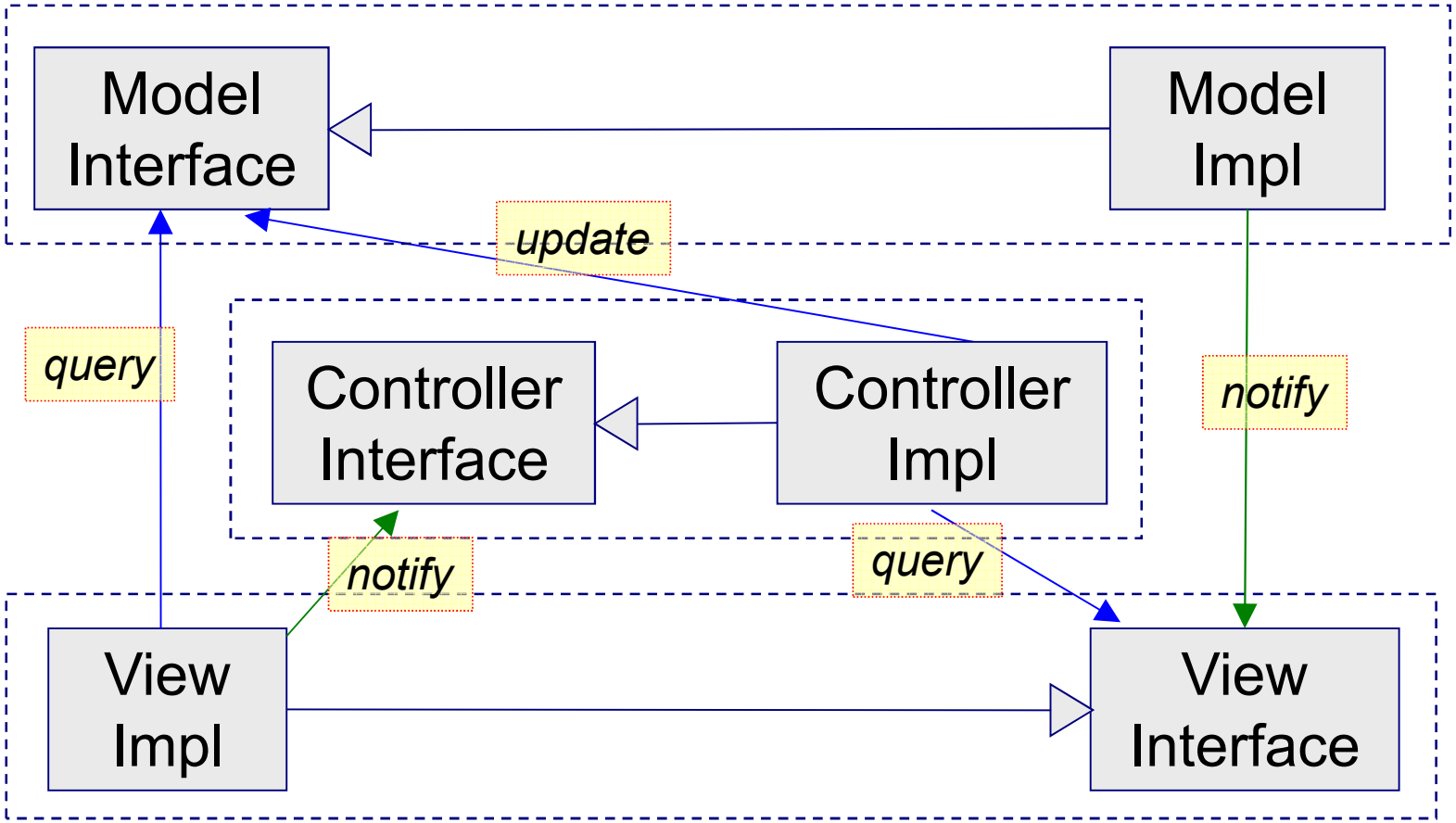
```
tf = new JTextField("", 20);
btm = new BinTreeModel();
JTree tree = new JTree(btm);
JScrollPane scrollPane = new JScrollPane(tree);
tf.addActionListener(this);
```

MVC vs MVP

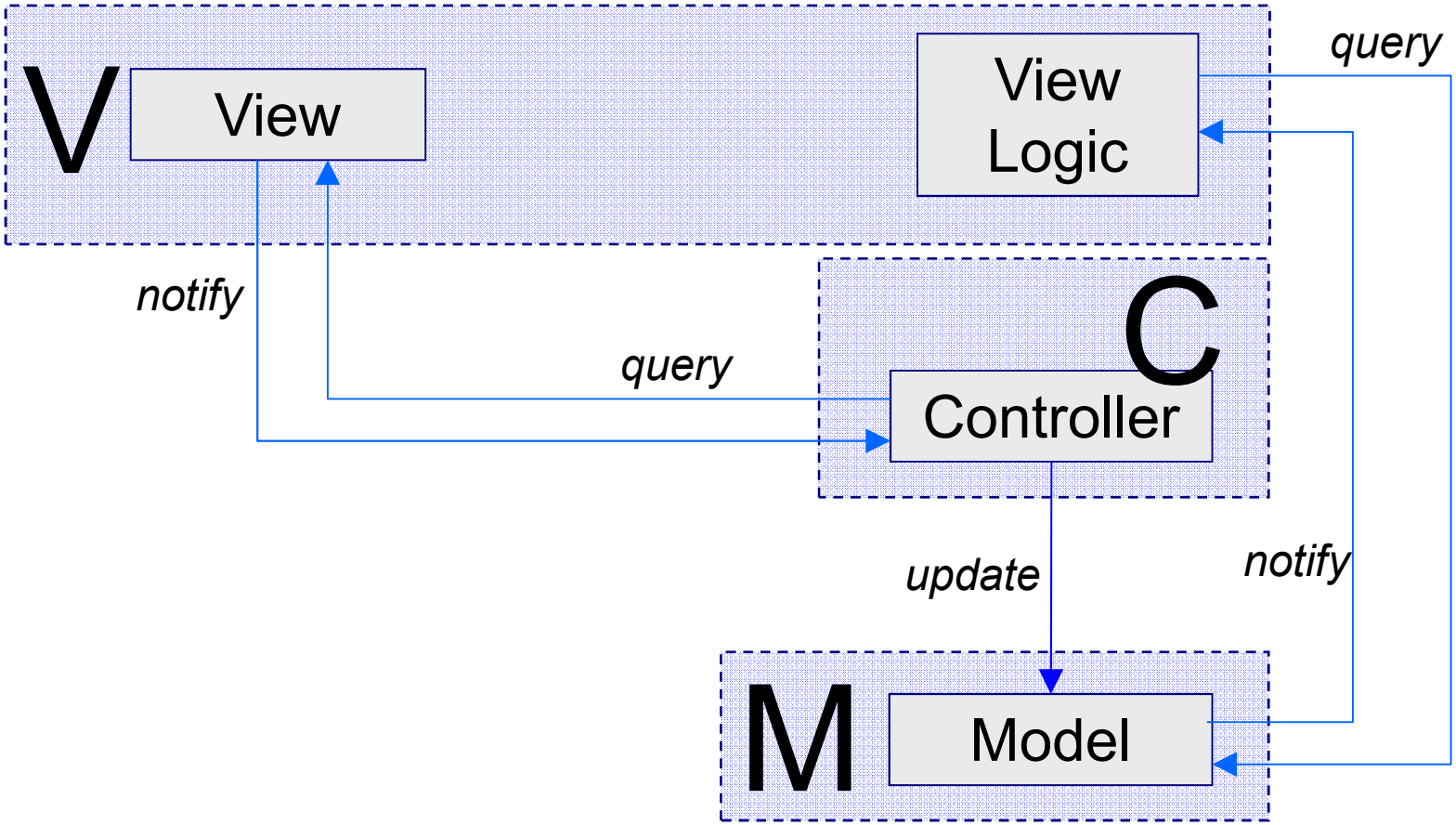
Questions of layering

- Thin vs thick client
  - where is the business logic?
    - MVC: how much business logic in *View*?
- Multi-tier architecture
  - browser, application server, database
  - display, service, business logic, infrastructure
- Smart browser
  - Ajax, GWT, HTML5, etc
    - browser does view *and model*
  - is it still thin?

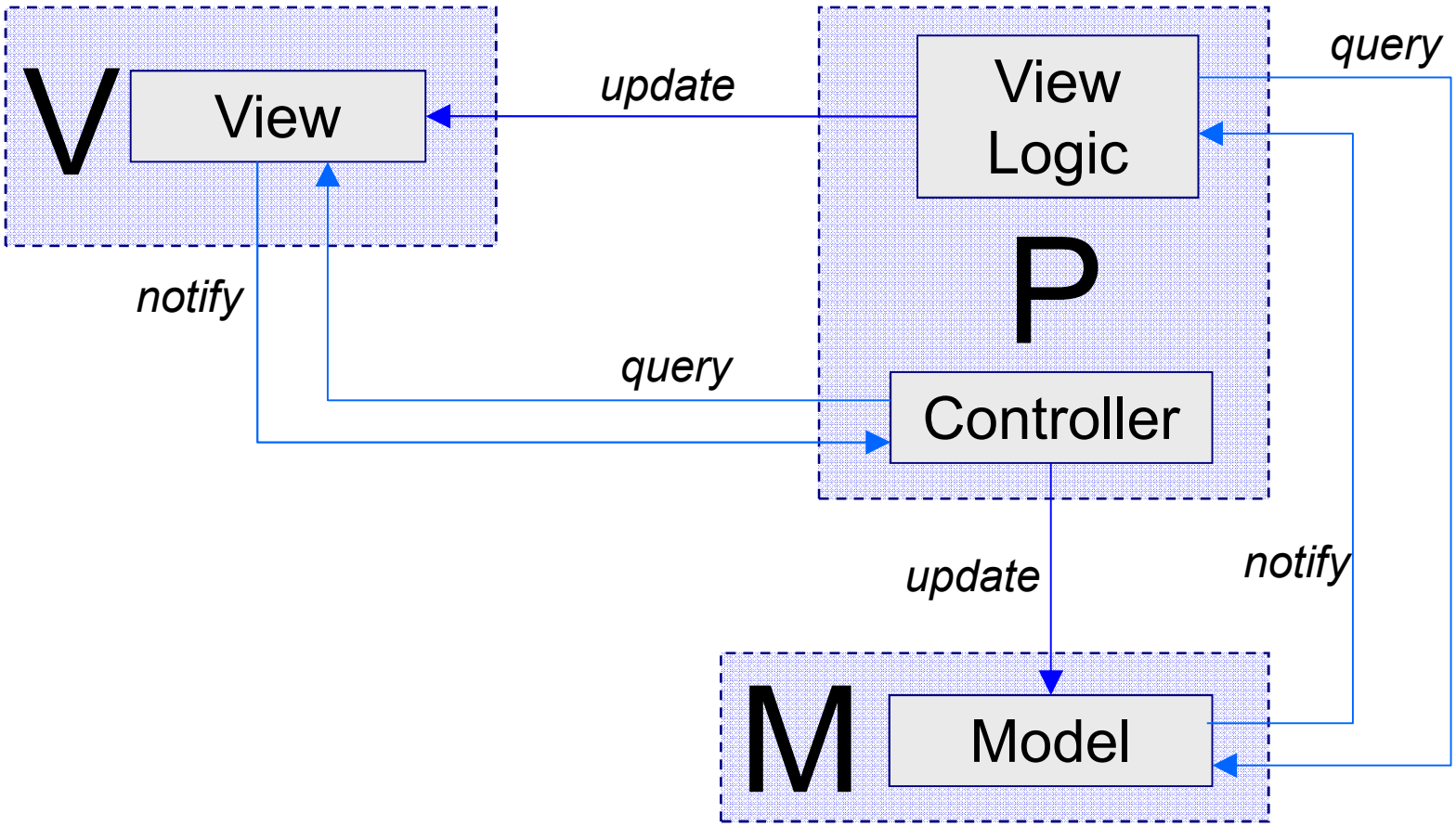
Classic MVC (revisited)



MVC vs. Model-View-Presenter



MVC vs. **Model-View-Presenter**



Low level graphics

JComponent and painting

- `JComponent`: basic class of widget hierarchy
  - paint(Graphics g)
    - paints on g when needed
  - repaint()
    - notifies GUI to repaint the component
    - variants with subrectangles specified
- `Graphics`: basic class for drawing primitives
  - the usual methods for drawing lines, arcs, text, etc
- `Graphics2d`: advanced graphics
  - shapes, glyphs, etc

Class `Graphics`

- Drawing primitives
  - *drawLine*, *drawRect*, *drawPolygon*, *drawText* etc.
- Image handling
  - *drawImage*, see class *Image* also
- Filling primitives
  - *fillRect*, *fillOval*, *fillPolygon*, etc.
- Clearing primitives
  - *clearRect*
- Colour and font setting
  - *setColor*, *setFont*, etc

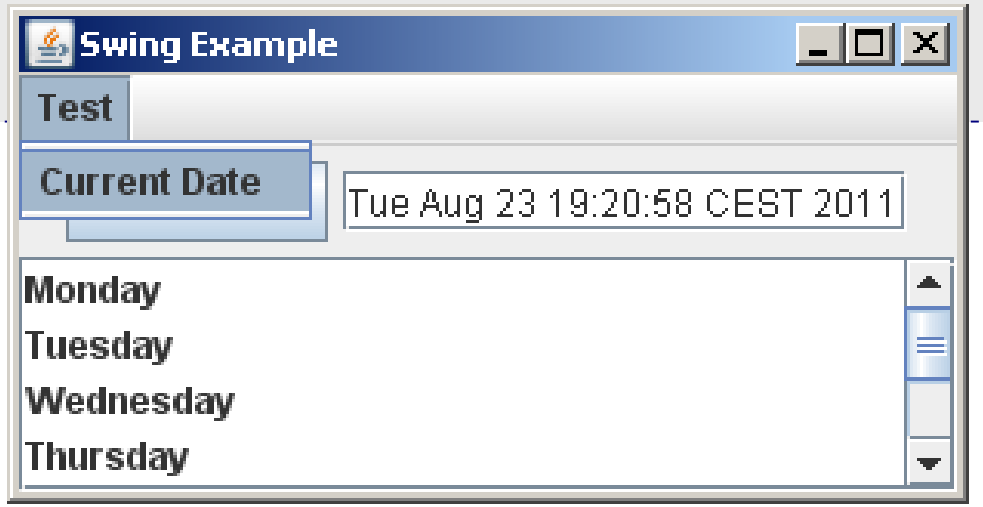
Accessing class Graphics

- Inside method `paint(Graphics g)`
  - method parameter
  - use only in the method
- Outside method `paint`
  - e.g. off-screen graphic rendering
  - `JComponent.getGraphics()` method
- Creating off-screen images (buffering)
  - use class `BufferedImage`
    - `java.awt.image` package

Special components

Adding menus to frames

```
ActionListener al = new MyActionListener();
b.addActionListener(al);
JMenuItem mi1 = new JMenuItem("Current Date");
mi1.setActionCommand("date"); // setting action command
mi1.addActionListener(al);    // adding listener
JMenu m1 = new JMenu("Test");
m1.add(mi1);
JMenuBar bar = new JMenuBar();
bar.add(m1);
f.setJMenuBar(bar);
```

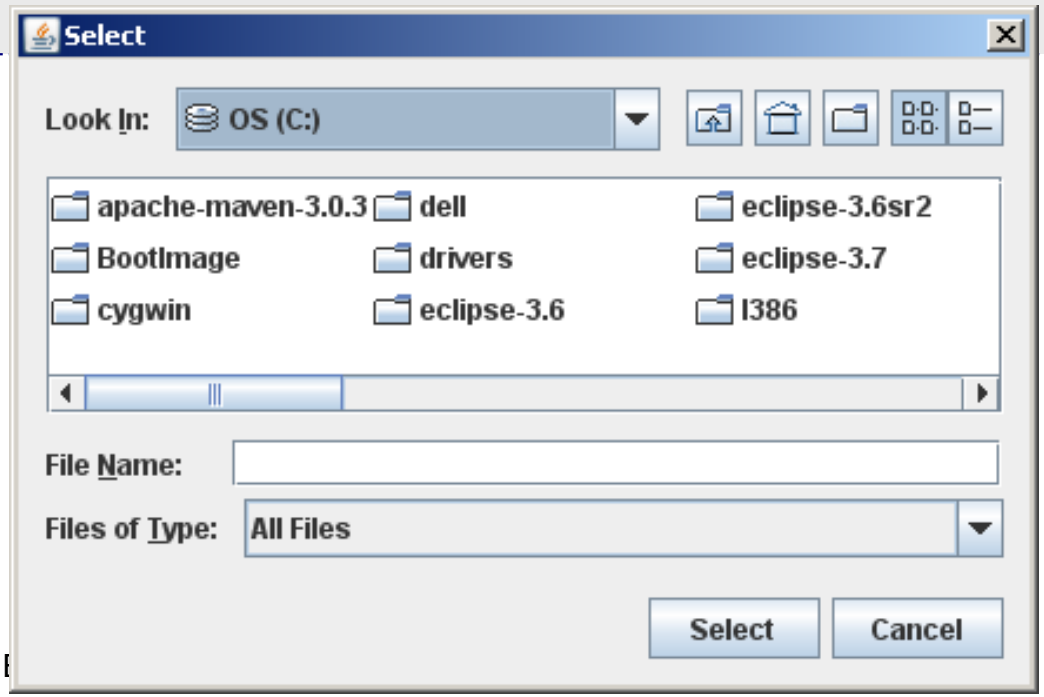


Using dialogs

- Dialogs are transient windows
- Default swing dialogs
  - JFileChooser
    - file selection; OK button text can be changed, file pattern can be set, etc
  - JColorChooser
    - for choosing a colour from a palette with different models
  - JOptionPane
    - selecting options (OK, OK-cancel, yes-no-cancel, etc.)
  - JDialog
    - general purpose empty dialog, has to be filled

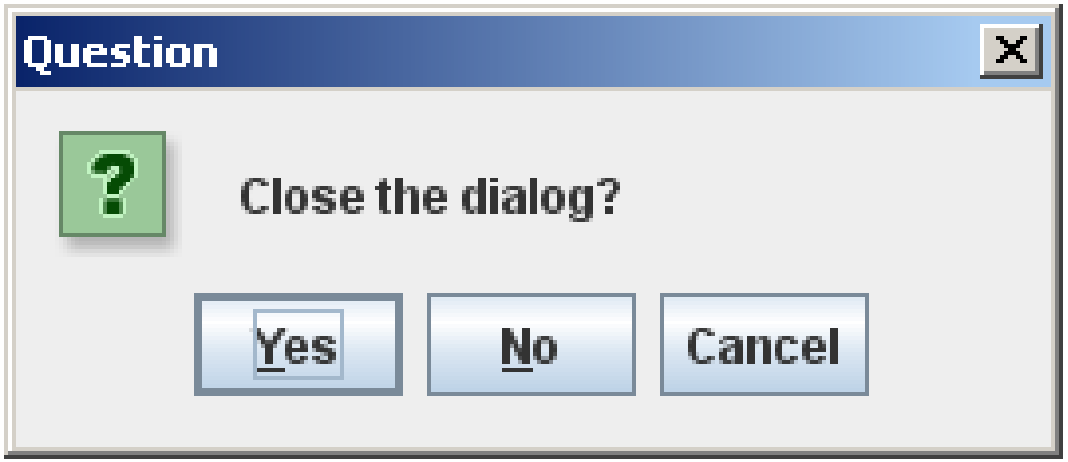
Dialog examples

```
JFileChooser chooser = new JFileChooser();
// parent window and approve text has to be defined
int returnVal = chooser.showDialog(f, "Select");
if(returnVal == JFileChooser.APPROVE_OPTION) {
       System.out.println(chooser.getSelectedFile()
              .getName());
}
```



Dialog examples

```
JOptionPane opt = new JOptionPane(“Close the dialog?",
      JOptionPane.QUESTION_MESSAGE, // type
      JOptionPane.YES_NO_CANCEL_OPTION); // options
// parent and dialog title
JDialog jd = opt.createDialog(f, "Question");
jd.setVisible(true);
System.out.println(opt.getValue().toString());
```



Further features

Further swing features

- Convenient tooltip management
  - e.g. *JComponent.setToolTipText*
- Inside windows
  - within frames with own window management
- Drag and drop
  - text and other objects, works across applications
- Look and feel
  - customizable by adding a simple jar file
- ...