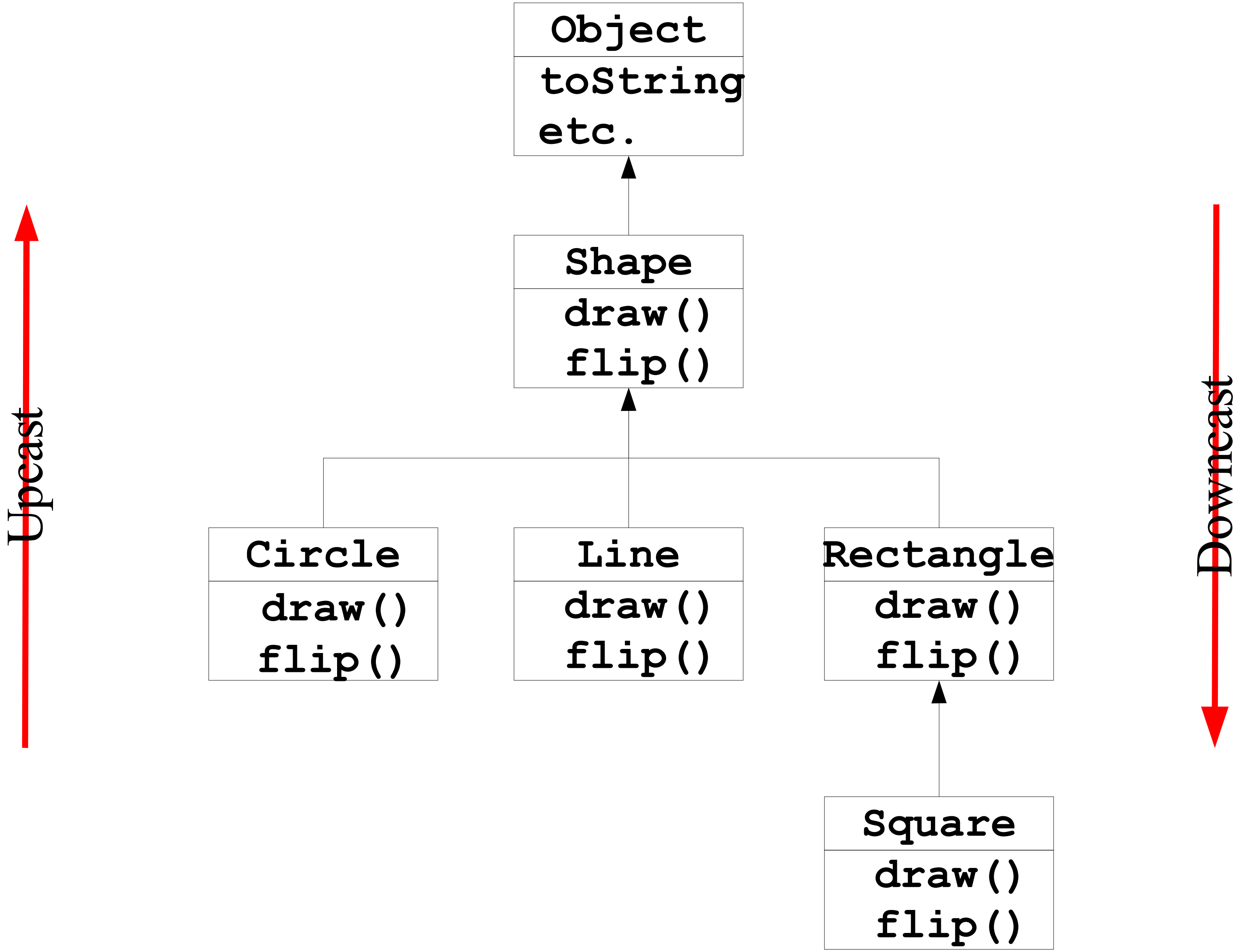
Reflection and JavaBeans

- Run-Time Type Identification (RTTI)
- Reflection
- The **`java.lang.Class`** class
- JavaBeans
    - Component-based development in Java

The Shape Hierarchy



The Shape Hierarchy, cont.

```
package figures;

public class Circle extends Shape {
     static {
          System.out.println ("Loading Circle");
     }
    public void draw() {System.out.println ("Circle");}
    public void flip() {System.out.println ("Ci.flip");}
}


public class Square extends Rectangle {
     static {
          System.out.println("Loading Square");
     }
    public void draw() {System.out.println ("Square"); }
    public void flip() {System.out.println ("Sq.flip");}
}

// similar implementations for the remaining classes
```

The Shape Hierarchy, cont.

```
import java.util.*;     // for ArrayList
import figures.*;       // for Shape hierarchy

public class UseShapes{
   public static void main (String args[]){
      ArrayList al = new ArrayList();
      al.add (new Circle()); al.add (new Line());
      al.add (new Rectangle());al.add (new Square());

      Iterator i = al.iterator();
      while (i.hasNext()){
         Shape s = (Shape)i.next(); // partial downcast
         s.flip(); // uses dynamic binding
      }
   }
}
```

- Shape downcast basic form of using RTTI
- All downcast are dynamic, i.e., checked at run-time, exception may be thrown (very different from C++).

Run-Time Type Identification (RTTI)

- When you need to know the exact type of a generic reference

- Can be used to special expensive methods
  - e.g., flip for a Square
- Has a method that only works for specific types
  - Paint all lines blue and circles red
- Do not overuse it

The `java.lang.Class` Class

- There is a **`Class`** object for each class in your system.
- The way Java type information is represented at run-time.
- Information for **`Class`** object is stored in a .class file
  - Load when first object is created or static access
- The **`Class`** object is used to create all the objects of that class.

- The Java Virual Machine (JVM) finds the appropriate .class file and loads it as a **`Class`** object the first time you *need* that class.
  - Goes through the directories listed in the CLASSPATH

Class Loading Example

```java
import figures.*; // use Shape hierarchy

public class LoadOrder{
  public static void main(String[] args){
    System.out.println ("Before Circle");
     Circle c = new Circle (); // loads Circle.class

     System.out.println ("Before Line");
     try {
       Class.forName("Line"); // load Line.class
     } catch (ClassNotFoundException e){
         e.printStackTrace (System.err);
     }

    System.out.println ("Before Square");
     // loads Rectangle.class and Square.class
     Square s = new Square();
}  }
```

- Java program loaded as needed, i.e., not all classes loaded at startup of program. (smart in network?)

RTTI Overview

- RTTI is performed with the **`Class`** object
- Must have a reference to the **`Class`** object
  - **`Class.forName(„Line”)`**
  - **`java.lang.Object`** has **`getClass()`** method that returns a **`Class`** object
- **`Class`** has the following methods
  - getName( )
  - getSuperclass( )
  - isInterface( )
  - isPrimitive( )
  - getMethods()
  - getFields()
  - getConstructors()
  - getPackage()
  - Plus many more

Three Types of RTTI

- The dynamic explicit downcast
  - `Shape s = (Shape)i.next();`
  - Uses RTTI to check that downcast is correct, may throw unchecked exception **`ClassCastException`**.
- Use the **`Class`** object
  - `Class.forName("Line")`
  - Can be queried at runtime for various information
- Use the **`instanceof`** keyword

  -
    ```
    if (o instanceof Car){ // equals() method
      Car c = (Car)o;
      /* check fields equal */
    }
    else {
      return false;
    }
    ```

Java Class Literals

- Introduced in Java 1.1
- **`Line.class`** instead of **`Class.forName(„Line“)`**
  - Must know that class exists at compile time
  - Checked at compile time (more safe to use)
  - No try block (fewer lines of code)
  - Faster
- Checked at compile time; no try block necessary, faster.

Reflection Overview

- RTTI deals with types you know of at compile time.
- Reflection deals class that you all get to know at run-time
- Useful in
  - JavaBeans
  - Remote Method Invocation (RMI)

- Two Examples
  - Class sniffer learn the details of a class at run-time
  - An optimization flip on the Shape hierarchy using reflection

Reflection, Examples

```
/** Gets the package name */
public static void getPackageName (Object o){
    Class c = o.getClass();
    Package p = c.getPackage();
    String packageName = "<default>";
    if (p != null) {
        packageName = p.getName();
    }
    System.out.println ("Package name : " + packageName);
}
```

```
/** Finds the class name of an object */
public static void getClassName (Object o){
    Class c = o.getClass();
    String className = c.getName();
    System.out.println ("Class name :" + className);
}
```

Reflection, Examples, cont.

```
import java.lang.*;          // for Class class
import java.lang.reflect.*;  // for reflection capabilities
import java.util.*;

/** Finds the method names defined on a object */
public static void getMethods (Object o){
  Class c = o.getClass();
  Method m[] = c.getMethods();
  for (int i = 0; i <m.length; i++){
     Method met = m[i];
     String methodName = met.getName();
     System.out.println ("  " + methodName);
     }
}
```

- For more details see the package **`java.lang.reflect`**.

Optimizing using Reflection

```
import java.util.*;
import figures.*;

public class OptimizedFlip{
   public static void main (String args[]){
      ArrayList al = new ArrayList();
      al.add (new Circle()); al.add (new Line());
      al.add (new Rectangle());al.add (new Square());
      Iterator i = al.iterator();
      while (i.hasNext()){
         Shape s = (Shape)i.next();
         //if (s instanceof Square || s instanceof Circle){
         Class c = s.getClass();
         // optimized cases
         if (c.getName().equals("figures.Square")
             || c.getName().equals("figures.Circle")){
                System.out.println("flip() optimized out");
         }
         else {
            s.flip(); // default behaviour
}}}}
```

JavaBeans

- Philsophy: Software as Lego bricks.
- A JavaBean is simply a Java class
  - i.e., no special language extension to support JavaBeans

- Main challange: to discover properties and events of existing JavaBean components.
- Uses reflection to dynamically investigate the components to discover which properties they support.
- Mainly for visual programming.

JavaBean Parts

- Properties
  - To be able to customize the JavaBean.
- Events
  - i.e., to allow the JavaBean to be interconnected to other JavaBean components or other parts of the application.
- Persistence
  - To allowed tools (e.g., GUI builders) to load and store JavaBeans in a standard way (using the **`Serializable`** interface)

JavaBeans Naming Convention

- For a property **`xyz`** og type **`T`** make two methods
  - **`public T getXyz()`**
  - **`public void setXyz(T newXyzValue)`**
- Alternatively for a boolean property **`bool`**
  - **`public boolean getBool()`**
  - **`public void isBool(boolean newBoolValue)`**

- For a listener MyListener
  - **`public void addMyListener (MyListener m)`**
  - **`public void removeMyListener (MyListener m)`**

- Other methods are „plain“ methods of the JavaBean

Other JavaBean Conventions

- A class implementing a JavaBean must have a default constructor.
  - To be able to construct objects of the type
- A JavaBean must implement the **`Serializable`** interface
- Jar file
  - A JavaBean must be stored in a jar file.

JavaBeans Example

```
import java.awt.*;
import java.awt.event.*;
import java.io.*; // to be able to implement serializable
public class Car implements Serializble{
    private String make;
    private String model;
    private double price;
    public Car() { this ("", "", 0.0); } // def. constructor

    public Object clone() {
        return new Car (this.make, this.model, this.price);
    }
    // JavaBeans Methods
    public String getMake() { return make; }
    public void setMake (String m) { make = m; }

    public String getModel() { return model; }
    public void setModel (String mo) { model = mo; }

    public double getPrice() { return price; }
    public void setPrice(double newPrice) { price = newPrice; }
}
```

JavaBeans Example, cont.

- A mainfest file must be stored with the Java class(es) that implements a JavaBean, for the Car example a file name, Car.mf.

```
Manifest-Version: 1.0

Name: bean/Car.class
Java-Bean: True
```

- Make a Jar with the supplied jar tool

```
>jar cfm Car.jar Car.mf bean
```

JavaBean Developer Kit (BDK)

- To help you in building JavaBeans

- Freely available from java.sun.com
- Documentation
  - Examples (including source code)
  - Reference manuals
- BeanBox
  - An application for testing JavaBeans.

- Demo of the BeanBox

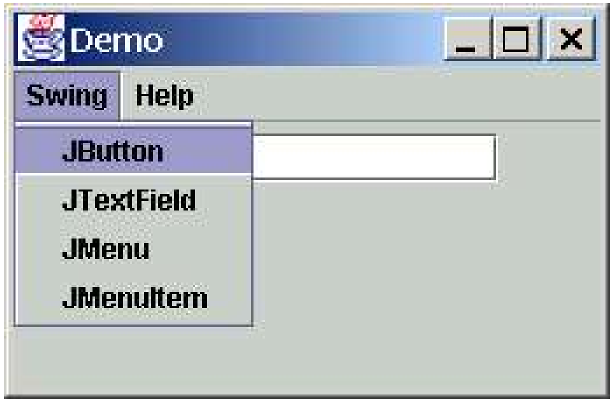
Summary

- Run-time Type identification used for downcast.

- With the **`java.lang.reflect`** package the details of a „unknown“ class can be explored at run-time.

- The reflection capablities of Java is used extensively in JavaBeans.
- JavaBeans is a way to build plug-and-play visual components in Java.
- JavaBeans not supported by BlueJ

Menu and Menu Items



- The class **`JMenuBar`**, **`JMenu,`** and **`JMenuItem`** are used for this purpose.

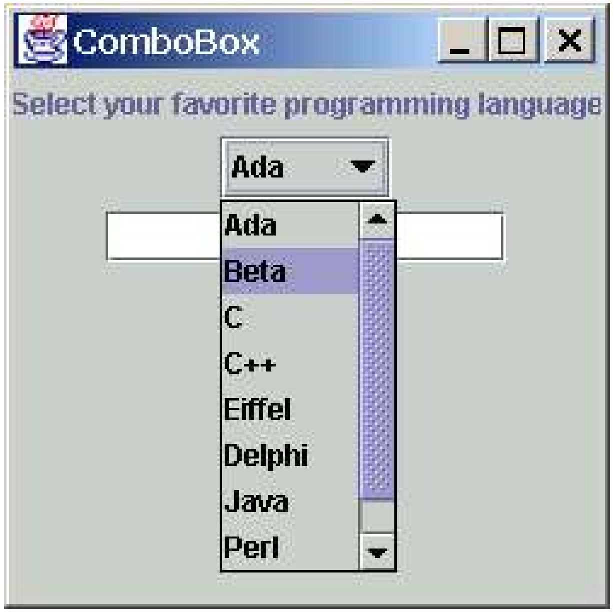
Menu and Menu Items, cont.

```
public class DemoApplet extends JApplet {
  JTextField t = new JtextField(15);
  Container cp;
  // use anonymous inner class
  ActionListener al = new ActionListener() {
    public void actionPerformed(ActionEvent e){
      t.setText(((JMenuItem)e.getSource()).getText());
    }
  };


  JMenu[] menus = { new JMenu("Swing"),
                    new JMenu("Help")};

  JMenuItem[] swingItems = { new JMenuItem("JButton"),
                             new JMenuItem("JTextField"),
                             new JMenuItem("JMenu"),
                             new JMenuItem("JMenuItem")};

  JMenuItem[] helpItems = { new JMenuItem("Topics"),
                            new JMenuItem("About") };
```

Menu and Menu Items, cont.

```
public void init() {
    // the swing menu
    for(int i = 0; i < swingItems.length; i++) {
      swingItems[i].addActionListener(a1);
      menus[0].add(swingItems[i]);
    }
    // the help menu
    for(int i = 0; i < helpItems.length; i++) {
      helpItems[i].addActionListener(a2);
      menus[1].add(helpItems[i]);
    }
    // create the menu bar
    JMenuBar mb = new JMenuBar();
    for(int i = 0; i < menus.length; i++) {
      mb.add(menus[i]);
    }
    // set up the menu bar
    setJMenuBar(mb);
    cp = getContentPane();
    cp.setLayout(new FlowLayout());
    cp.add(t);
  }
```

Combo Box



- The class **`JComboBox`** is used for this purpose.
- One and only one element from the list can be selected.

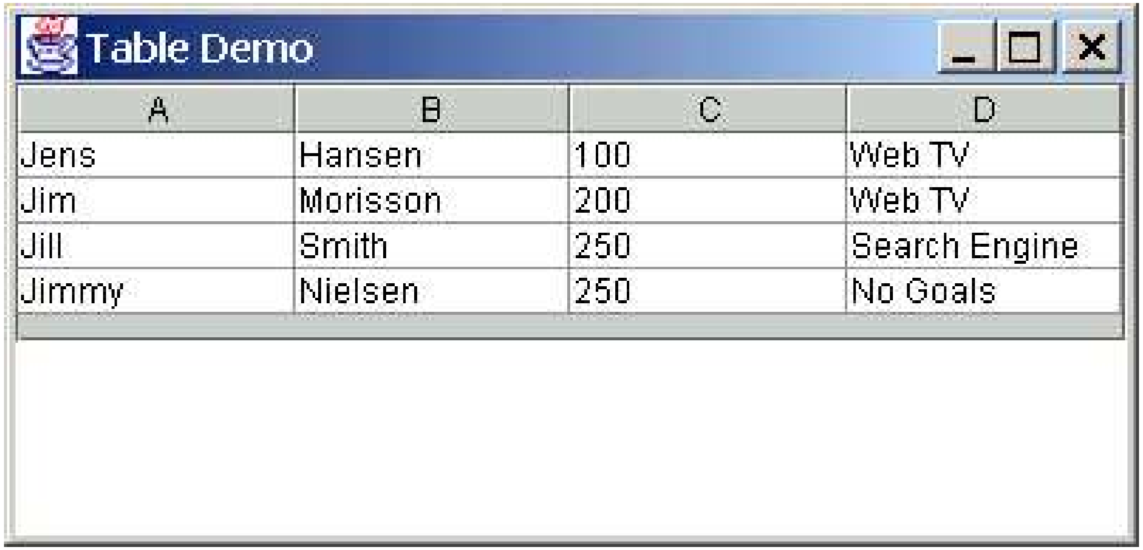
Combo Box, cont.

```
public class ComboBox extends JApplet {
  JTextField t = new JTextField(15);
  JLabel     l =
    new JLabel ("Select your favorite programming language");
  Container cp;

  ActionListener al = new ActionListener() {
    public void actionPerformed(ActionEvent e){
      t.setText(
       (String)((JComboBox)e.getSource()).getSelectedItem());
    }
  };

  String[] languages = { "Ada", "Beta", "C", "C++",
                         "Eiffel", "Delphi", "Java",
                         "Perl", "Python"};
  JComboBox cb = new JComboBox();
```

Combo Box, cont.

```
public void init() {
 // populate the combo box
 for(int i = 0; i < languages.length; i++) {
    cb.addItem(languages[i]);
  }
  // connect the action listener
  cb.addActionListener (al);
  cp = getContentPane();
  cp.setLayout(new FlowLayout());
  cp.add(l);
  cp.add(cb);
  cp.add(t);
}
public static void main(String[] args) {
  ComboBox applet = new ComboBox();
  JFrame frame = new JFrame("ComboBox");
  frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
  frame.getContentPane().add(applet);
  frame.setSize(250,250);
  applet.init();
  applet.start();
  frame.setVisible(true);
```

Tables



A	B	C	D
Jens	Hansen	100	Web TV
Jim	Morisson	200	Web TV
Jill	Smith	250	Search Engine
Jimmy	Nielsen	250	No Goals

- The classes **`JTable`** and **`AbstractTableModel`** are used.
  - The latter controls the data

Tables, cont.

```
public class Table extends JApplet {
  JTextArea text = new JTextArea(4, 24);

  // AbstractTableModel controls all data
  class TModel extends AbstractTableModel {
    Object[][] table_data = {
      {"Jens",  "Hansen",   "100", "Web TV"},
      {"Jim",   "Morisson", "200", "Web TV"},
      {"Jill",  "Smith",    "250", "Search Engine"},
      {"Jimmy", "Nielsen",  "250", "No Goals"}};

    // reprint table data when changes
    class TMList implements TableModelListener {
      public void tableChanged(TableModelEvent e){
        text.setText(""); // clear screen
        for(int i = 0; i < table_data.length; i++) {
          for(int j = 0; j < table_data[i].length; j++){
            text.append(table_data[i][j] + " ");
          }
          text.append("\n");
        }      }    }
```

Tables, cont.

```
  public TModel() {
    addTableModelListener(new TMList());
  }
  public int getColumnCount() {
    return table_data[0].length;
  }
  public int getRowCount() {
    return table_data.length;
  }

  public Object getValueAt(int row, int col) {
    return table_data[row][col];
  }
}
public void init() {
  Container cp = getContentPane();
  JTable the_table = new JTable(new TModel());
  cp.add(the_table);
  cp.add(BorderLayout.CENTER, text);
}
```