1 Iterators

An iterator object is for moving through a (any) collection. Iterator is a Java interface specifying methods for moving through a collection. It takes a type parameter E and requires the following methods.

- boolean hasNext()
- E next() (It returns the next element in the collection, updating the position in the collection.)
- void remove() (It removes from the collection the last element returned by next.)

We have been writing for-each loops for some time. In fact, the for-each is understood as a loop involving iterators.

Example:

```java
private static double sum(List<Double> dList){
    double total = 0.0;

    for(Double d: dList){
        total = total + d;
    }

    return total;
}
```

The code below explicitly shows the iterator.

```java
private static double sum(List<Double> dList){
    double total = 0.0;

    for(Iterator<Double> it = dList.iterator(); it.hasNext();){
        Double d = it.next();
        total = total + d;
    }

    return total;
}
```

Further, it is possible to use iterators to accomplish tasks that a for-each loop cannot. In particular, iterators can be used to move through several lists at once.

Example:
private static ArrayList<Double> sum(List<Double> dL1, List<Double> dL2){
    ArrayList<Double> result = new ArrayList<Double>();

    Iterator<Double> it1 = dL1.iterator();
    Iterator<Double> it2 = dL2.iterator();

    while (it1.hasNext() && it2.hasNext()){
        Double d1 = it1.next();
        Double d2 = it2.next();

        result.add(d1+d2);
    }

    return result;
}

Lists also have a method for returning an extension of Iterator: ListIterator. It has some additional methods such as previous which can be used to go backwards in a list.

2 Comparable and Comparator

Many elements that we might want to put into a collection have a natural ordering. To indicate that objects in a particular class are totally ordered, the class implements the Comparable interface, which requires a single method: compareTo.

If we had to write the Comparable interface, it would look like the following.

```
public interface Comparable<E>{
    int compareTo(E o);
}
```

The method compareTo returns a negative number if this < o, 0 if this = o, and a positive number if this > o.

The classes Integer and Double implement Comparable.

Example:

```
Integer i = 5;
Integer j = 7;

System.out.println(i.compareTo(j)); //prints -1
```

Further, there is a class called Collections that has a static sort method that can be used on lists of comparable objects.

Example:
List<Double> list = new ArrayList<Double>();
...
Collections.sort(list);

However, we might want to sort the list using a different way of comparing objects. The most natural thing to do would be to pass in to sort a different comparison method; however, methods are not first class objects in Java. Since we can’t pass a method, we do the next best thing, and pass an object that contains the method. There is a standard interface to implement: Comparator.

Given an element type E, the Comparator interface requires the method int `compare(E o1, E o2)`. Example:

```java
import java.util.*;

public class MyDoubleComparator implements Comparator<Double>{
    public int compare(Double o1, Double o2){
        return o2.compareTo(o1); //opposite order
    }
}
...

public static void main(String[] args){
    ...  
    List<Double> list = new ArrayList<Double>();
    ...
    MyDoubleComparator c = new MyDoubleComparator();
    Collections.sort(list, c); //sort in descending order
    ...}
```

3 Introduction to the Java GUI Libraries

Unlike many languages, Java came with a high level library for creating Graphical User Interfaces (GUIs). The original library is called the Abstract Windows Toolkit (AWT), and

---

1. An object is said to be first class, if it can be stored in a variable, stored in a data-structure, passed into a method, and returned from a method.
it is still available in the java.awt package. This library worked by using the platform specific GUI to implement various classes. GUI components that are implemented this way are known as *heavyweight components*. This approach entailed that Java would have bugs and/or quirks that depended on the platform. Java designers wanted more control of the GUI, so they created a new implementation that relied on just a few simple platform specific components, and the rest was built on top of this core using Java itself. The parts implemented in Java are called *lightweight components*. The new GUI library’s name is Swing; it’s available in javax.swing. The convention is that the names for Swing components start with a ‘J’. Everyone has been encouraged to use Swing rather than AWT; however, it is not always possible.

4 Creating GUI Objects

4.1 Creating a Window

Use JFrame to create a window. The class JFrame has the following methods.

- JFrame(String title) The constructor.
- void setSize(int width, int height)
- void setLocation(int x, int y)
- void setLocationRelativeTo(Component c) centers the frame relative to c. If c is null, then the screen is used.
- void setVisible(boolean visible)
- void setDefaultCloseOperation(int mode) The default is JFrame.EXIT_ON_CLOSE which makes the window go away, but the program does not halt.
- Component add(Component c)
- void setLayout(LayoutManager m)

Example:

```java
import javax.swing.*;
...
JFrame frame = new JFrame("My Title");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
...
```
4.2 Creating a Button

Use JButton to create a button. The constructor is JButton(String label). Example:

```java
import javax.swing.*;
...
JFrame frame = new JFrame("My Title");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JButton button = new JButton("Press Me");
frame.add(button);

frame.setVisible(true);
...
```

4.3 Creating a Text Field

Use JTextField to create a text field. The constructor is JTextField(int n), where n is roughly the number of characters the field is wide.
Example:

```java
import javax.swing.*;
...
JFrame frame = new JFrame("My Title");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JTextField field = new JTextField(10);
frame.add(field);

frame.setVisible(true);
...
```

![Image of a Swing application window](image.png)

### 4.4 Creating a Label

Use `JLabel` to create a label. The constructor is `JLabel(String labelText)`. Example:

```java
import javax.swing.*;
...
JFrame frame = new JFrame("My Title");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JLabel label = new JLabel("Number:");
frame.add(label);

frame.setVisible(true);
...
```
5  Layout Managers

So far we’ve been able to create various GUI objects of interest, but they look ugly and fill the entire window. One approach that some languages have taken is to require that the size and position in the window of each object be explicitly specified. The Java designers felt that such an approach would not work well when distributing the same code on different platforms, and instead developed a more abstract approach. Rather than specify exact position, one specifies an abstract layout that, for example, might indicate that the GUI objects should start at the top left of the window and be placed one after the other horizontally. The layouts are in the AWT library; thus Java code should include import java.awt.*; Each layout implements the Layout interface.

5.1 The Flow Layout

The FlowLayout class is for the example layout described about — albeit somewhat more general in that the alignment (left, right, or center) can be specified. There are a couple of constructors. The no-argument constructor defaults to center alignment. The one-argument constructor takes an integer alignment parameter: FlowLayout.LEFT, FlowLayout.RIGHT, or FlowLayout.CENTER.

Example:

```java
import javax.swing.*;
import java.awt.*;
...
JFrame frame = new JFrame("My Title");
LayoutManager layout = new FlowLayout(FlowLayout.LEFT);
frame.setLayout(layout);

JLabel label = new JLabel("Number:");
frame.add(label);
```
5.2 The Grid Layout

Instead of a flow across, one might want a table or a grid, where the grid fills the window. This is accomplished with the GridLayout class. One specifies the size of the grid when invoking the constructor (GridLayout(rows, cols)). To fill in the grid, just add; the grid entries are filled in from top to bottom, and left to right.

Example:

```java
import javax.swing.*;
import java.awt.*;
...

JFrame frame = new JFrame("My Title");
LayoutManager layout = new GridLayout(2, 3);
frame.setLayout(layout);
frame.add(new JLabel("Name:"));
frame.add(new JTextField(10));
frame.add(new JButton("Press Me"));
frame.add(new JLabel("Phone Number:"));
frame.add(new JTextField(10));
frame.add(new JButton("Press Me Too"));
```
5.3 The BorderLayout

The BorderLayout divides the window into five regions: north, south, east, west, and center.

Example:
import javax.swing.*;
import java.awt.*;
...
JFrame frame = new JFrame("My Title");
LayoutManager layout = new BorderLayout();
frame.setLayout(layout);

frame.add(new JLabel("North "), BorderLayout.NORTH);
frame.add(new JLabel("South "), BorderLayout.SOUTH);
frame.add(new JLabel("East "), BorderLayout.EAST);
frame.add(new JLabel("West "), BorderLayout.WEST);
frame.add(new JLabel("Center ", JLabel.CENTER)
        , BorderLayout.CENTER);

frame.setSize(300,300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
6 Panels as Sub-Containers

Panels don’t have a look of their own. Rather their purpose is for grouping. Other components can be placed inside to achieve the desired look.
Example:

```java
import javax.swing.*;
import java.awt.*;

public class TestGUI{
    public static void main(String[] args){
        JFrame frame = new JFrame("My Title");
        LayoutManager layout = new BorderLayout();
        frame.setLayout(layout);

        frame.add(new JLabel("Top Message", JLabel.CENTER), BorderLayout.NORTH);
        frame.add(new JLabel("OK "), BorderLayout.SOUTH);

        JPanel p1 = new JPanel();
        LayoutManager layout1 = new FlowLayout(FlowLayout.CENTER);
        p1.setLayout(layout1);
        p1.add(new JButton("Solve"));
        p1.add(new JButton("Clear"));
        p1.add(new JButton("Help"));

        JPanel p2 = new JPanel();
        LayoutManager layout2 = new FlowLayout(FlowLayout.CENTER);
        p2.setLayout(layout2);
        p2.add(new JTextField(20));
    }
}
```
```java
JPanel p3 = new JPanel();
LayoutManager layout3 = new GridLayout(3, 1);
p3.setLayout(layout3);
p3.add(p2);
p3.add(p1);
p3.add(new JTextArea("Some help here."));

frame.add(p3, BorderLayout.CENTER);
frame.setSize(300, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

7 Color

Color is a class in AWT that is used in the GUI and graphics libraries to specify color. It is possible to create a color by specifying its red, green, and blue components; however, it is easier to use the standard colors which are constants associated with the color class.

The color of a button (or any component) can be set with the method `setBackground`, and the color of the text inside with `setForeground`.

Example:
```java
JButton button = new JButton("Press Me");
button.setBackground(Color.GREEN);
button.setForeground(Color.RED);
frame.add(button);
```