

OBJECT ORIENTED TECHNIQUES USING JAVA(ACSE0302)

Unit: 5

**Course Details
(B.Tech 3rd Sem /2nd Year)**

**Dr. Mohammad Shahid
Department
Of
Computer Sc & Engineering**

- A GUI represents an application that has a visual display for the user with easy to use controls.
- A GUI generally consists of graphical components like windows, frames, buttons, labels, etc.
- We can use these components to interact with the system or even the outside world.
- Java provides many APIs and reusable classes using which we can develop GUI applications.

One of the oldest kits provided by Java is 'Abstract Windowing Toolkit" or AWT.

- All newer APIs or components like Swing; JavaFX, etc. are based on this AWT.
- The GUI contains a sequence of activities that also trigger some events that in turn execute some actions on invoking a component or part of a component like by clicking a button we trigger some actions.

- So a GUI application is a **framework consisting of graphical components** & events that can be triggered on these components and the actions that execute as a result of events trigger.

Framework:- Frameworks usually provide precompiled reusable classes and components that we can drag and drop in the drawing area and then associate the events and actions with these components.

Component

- Component is an abstract class that encapsulates all of the attributes of a visual component.
- All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component.
- It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting..

Container

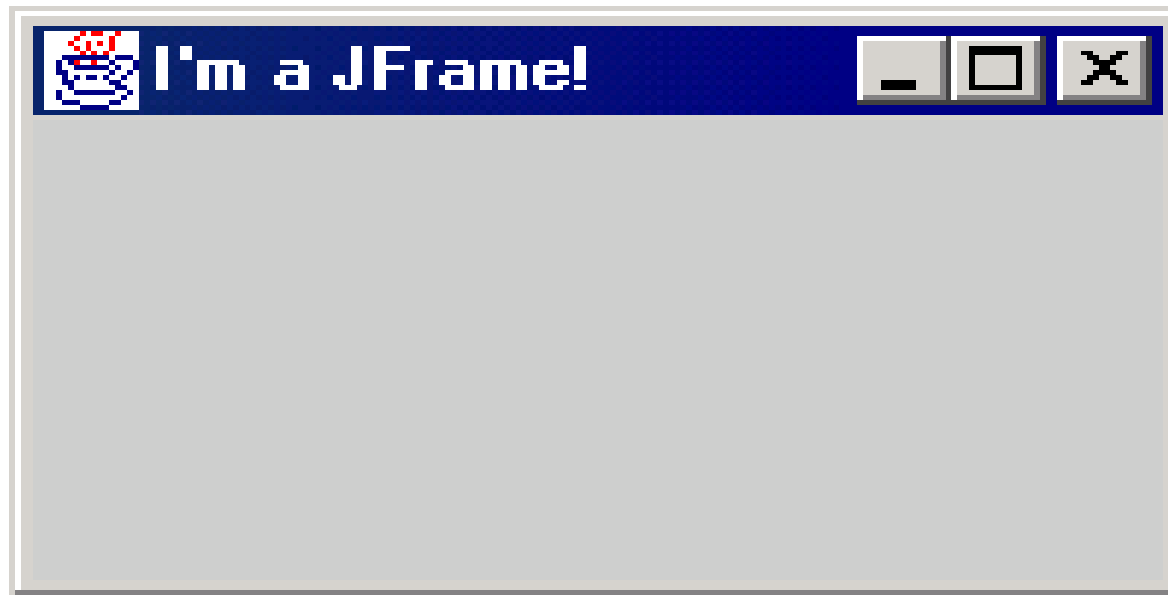
- The Container class is a subclass of Component.
- It has additional methods that allow other Component objects to be nested within it.
- Other Container objects can be stored inside of a Container (since they are themselves instances of Component).

Panel

- The Panel class is a concrete subclass of Container.
- Panel is a window that does not contain a title bar, menu bar, or border. This is why you don't see these items when an applet is run inside a browser.
- components can be by its **add()** method (inherited from Container).
- Added components position and resize manually using the **setLocation()**, **setSize()**, **setPreferredSize()**, or **setBounds()** methods defined by Component
- **Window class**
- The Window class creates a top-level window.
- It is not contained within any other object. it sits directly on the desktop.
- Window objects created using Window called Frame. A window does not have borders or menu bars.

Frame :-

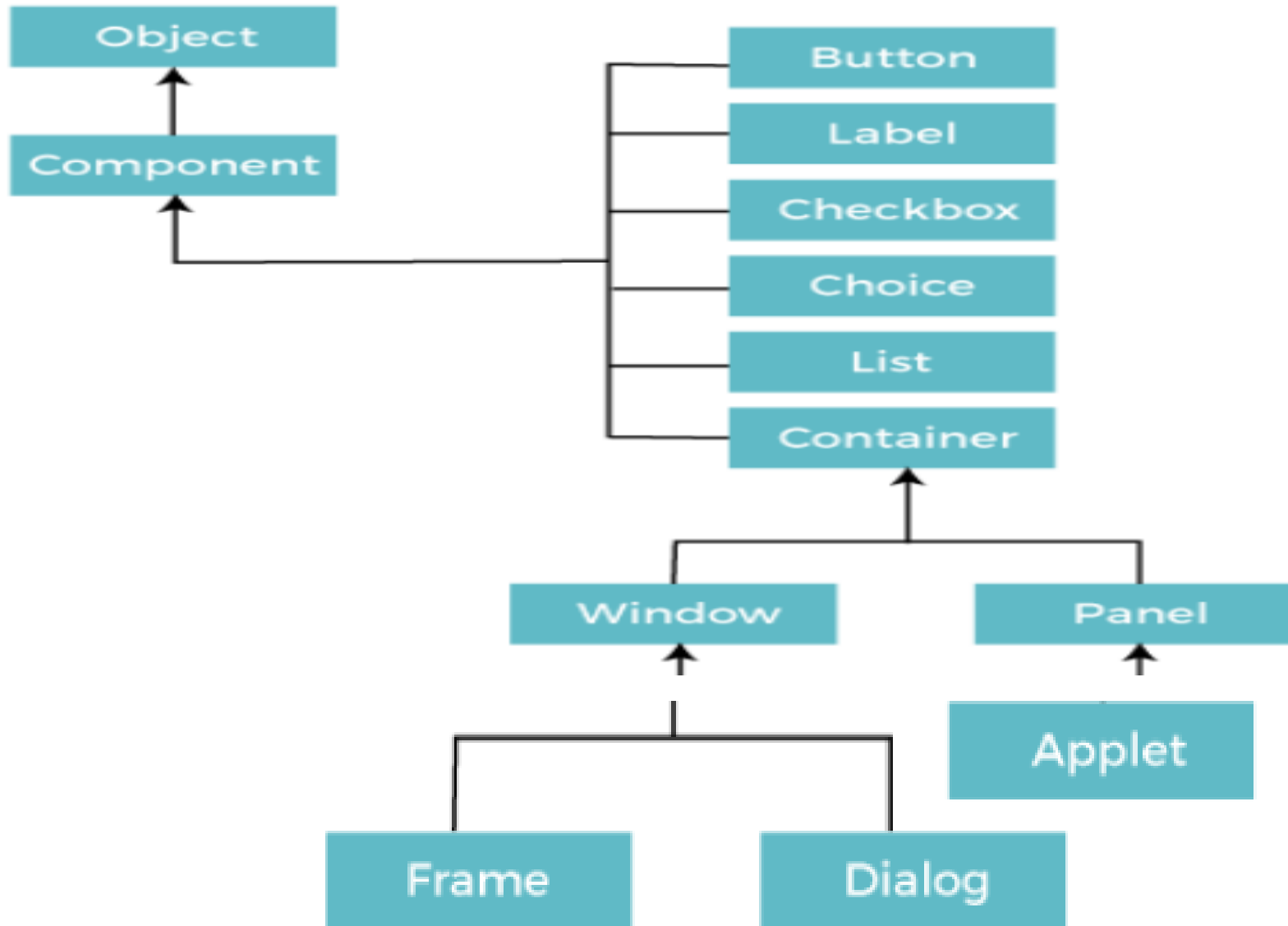
- It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners. If you create a Frame object from within an applet, it will contain a warning message, such as “Java Applet Window,” to the user that an applet window has been created.



Java AWT (Abstract Window Toolkit) is *an Java API to develop Graphical User Interface (GUI) or windows-based applications in Java.*

- It is a platform-dependent framework i.e. the GUI components belonging to AWT are not the same across all platforms.
- Due to its platform dependence and a kind of heavyweight nature of its components, it is rarely used in Java applications these days. Besides, there are also newer frameworks like Swing which are light-weight and platform-independent.
- Swing has more flexible and powerful components when compared to AWT. Swing provides components similar to Abstract Window Toolkit and also has more advanced components like trees, tabbed panels, etc.
- [Java Swing framework](#) is based on the AWT.

Java AWT Hierarchy



AWT Classes

AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East, West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.
Container	A subclass of Component that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.

Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for TextArea and TextField .
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

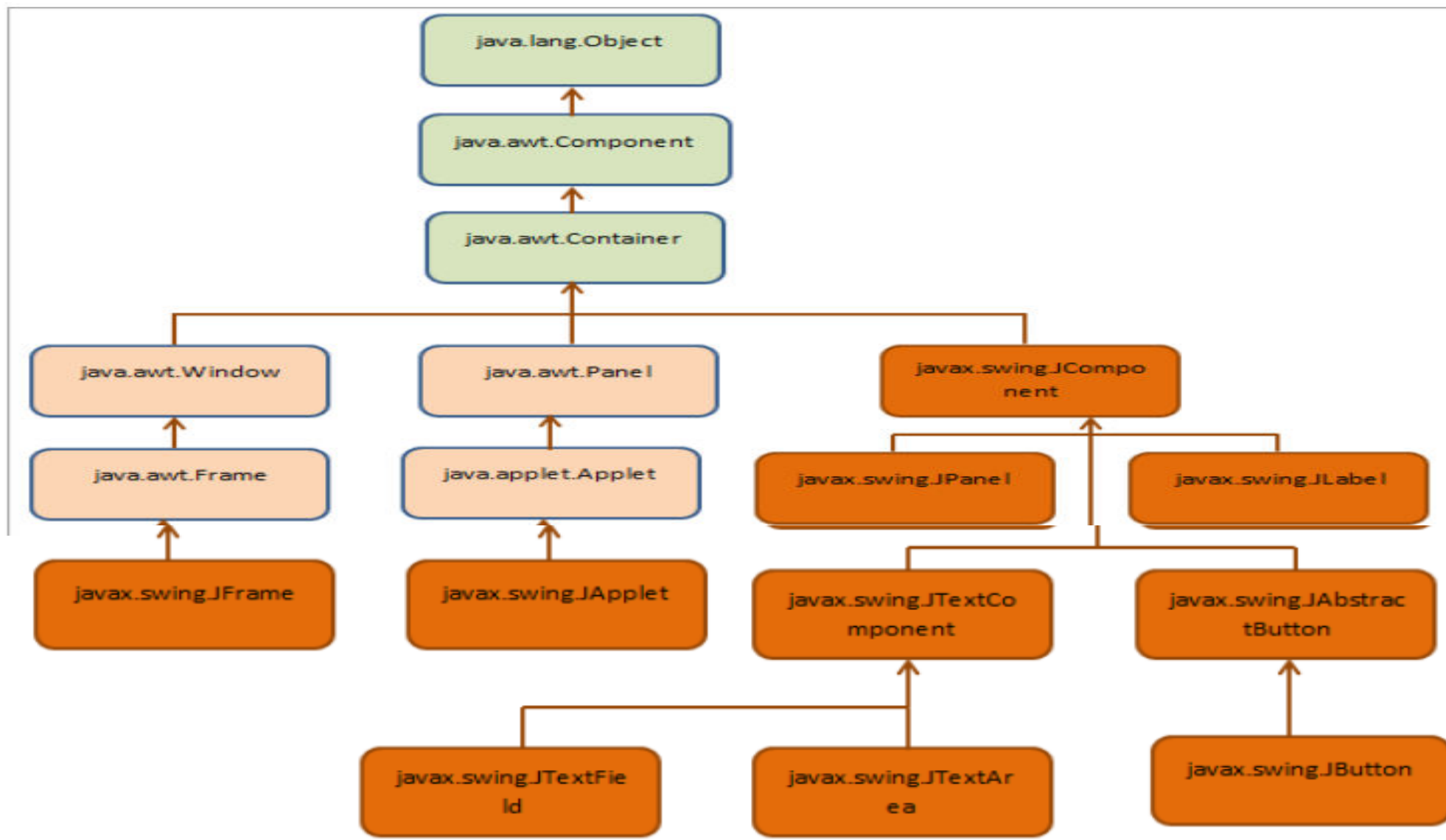
Java provides the following main frameworks.

Java provides the following frameworks for GUI programming:

- **Abstract Windowing Toolkit:** This is the oldest framework in Java and it was first introduced in JDK 1.0. Most of the AWT components are now outdated and are replaced by Java swing components.
- **Swing API:** This is a set of graphical libraries developed on top of the AWT framework and is a part of **Java Foundation Classes** (JFC). Swing has modular architecture wherein we can use plug-and-play for the components.
- **JavaFX:** The latest framework is available from Java 8 onwards.
- Java provides a set of features and functionality for developing graphical user interfaces or GUIs. This set of features is known as Java Foundation Classes or JFC. JFC contains classes from java.awt and javax.swing packages.

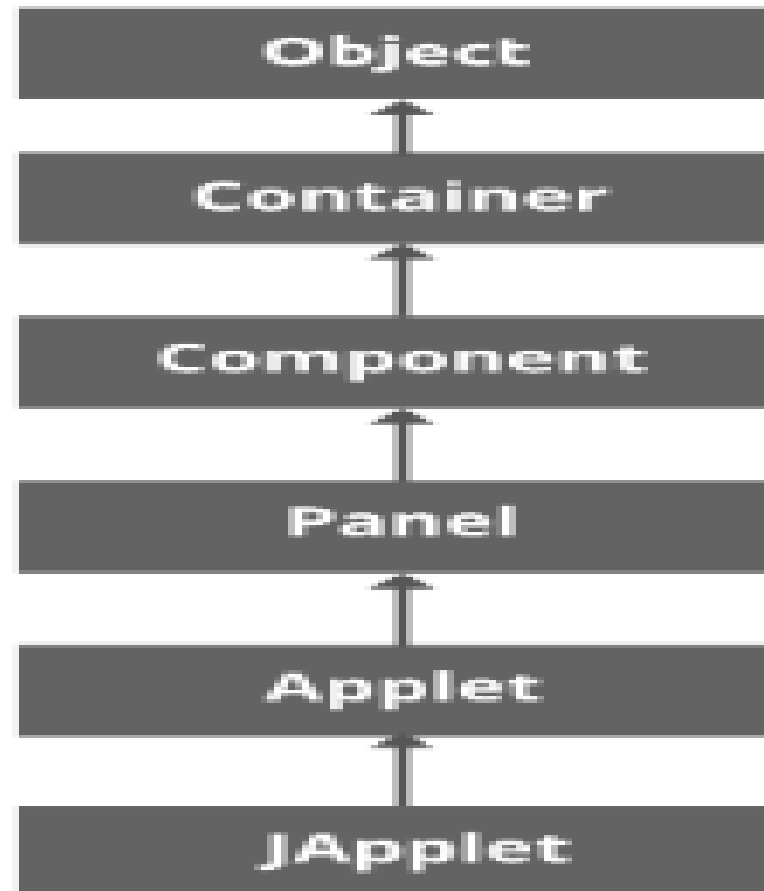
JFC Java Foundation Classes

The following diagram summarizes various components in JFC.



- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- All applets are sub-classes (either directly or indirectly) of java.applet.Applet class. Applets are not stand-alone programs. They run either within a web browser or an applet viewer.

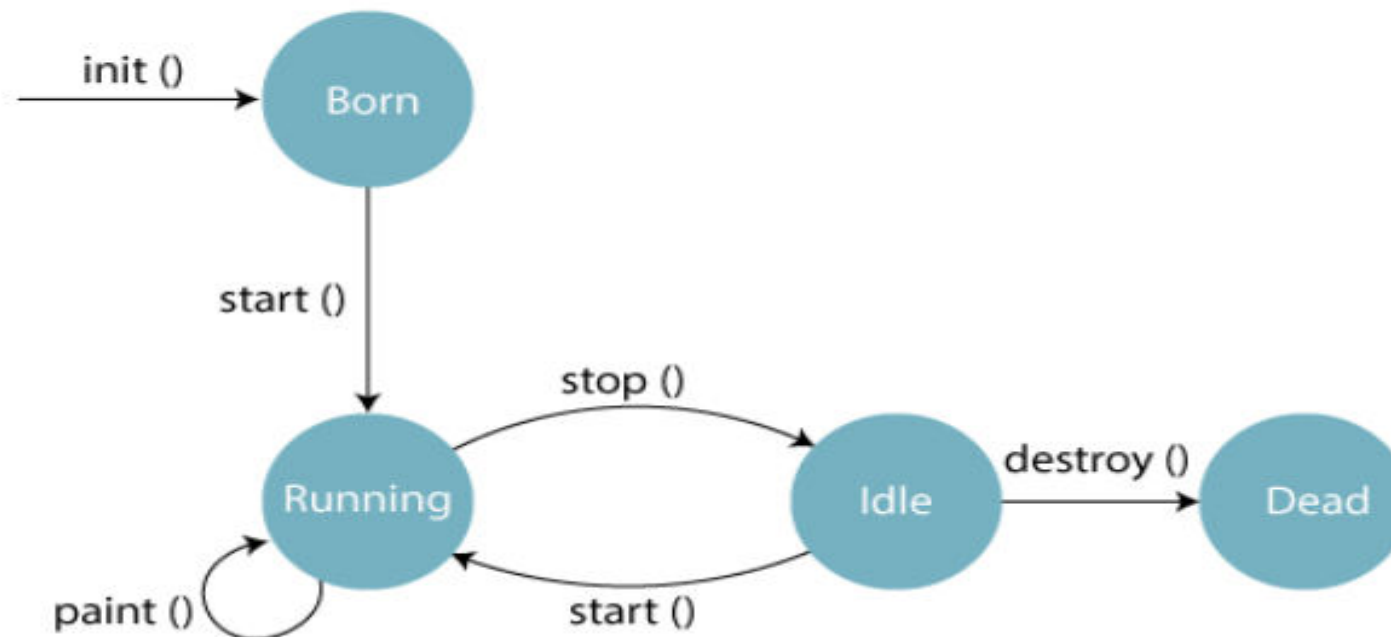
Applet Class Hierarchy



Life cycle for Applet

- The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely `init()`, `start()`, `stop()`, `paint()` and `destroy()`. These methods are invoked by the browser to execute.

Methods of Applet Life Cycle



Lifecycle methods for Applet:

- The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

- For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.
 1. **public void init():** is used to initialized the Applet. It is invoked only once.
 2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
 3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
 4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

- **The Component class provides 1 life cycle method of applet.**
 5. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How create & Run the Applet Program

Creating Applet

- java.applet.Applet class
- For creating any applet java.applet.Applet class must be inherited.

Who is responsible to manage the life cycle of an applet?

- Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool

Example 1 – Execute program by using HTML file

```
import java.applet.Applet;  
import java.awt.Graphics;  
public class FirstApplet extends Applet{  
    public void paint(Graphics g){  
        g.drawString("welcome",150,150);  
    }  
  
}
```

Save File :- FirstApplet.java

Compile :- javac FirstApplet.java

Run The Program by using HTML File

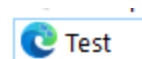
Create a new text File(in notepad)

```
<html>  
<body>  
<applet          code="FirstApplet.class"          width="300"  
    height="300">  
</applet>  
</body>  
</html>
```

Save File :- Test.html

Run :-

Click on Test



Example 1 – Execute program by using appletviewer

```
import java.applet.Applet;  
import java.awt.Graphics;  
public class FirstApplet extends Applet{  
    public void paint(Graphics g){  
        g.drawString("welcome",150,150);  
    }  
}
```

Save File :- FirstApplet.java

Compile :- javac FirstApplet.java

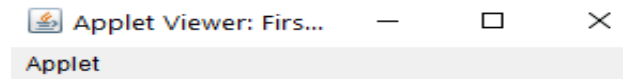
Run The Program by using HTML File

Create a new text File(in notepad)

```
<html>  
<body>  
<applet          code="FirstApplet.class"          width="300"  
    height="300">  
</applet>  
</body>  
</html>
```

Save File :- Test.html

Run :- appletviewer Test.html



welcome

Applet started.

Displaying Graphics in Applet

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example 2 – Graphics Program

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsDemo extends Applet{
    public void paint(Graphics g){
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);
        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);
    }
}
```

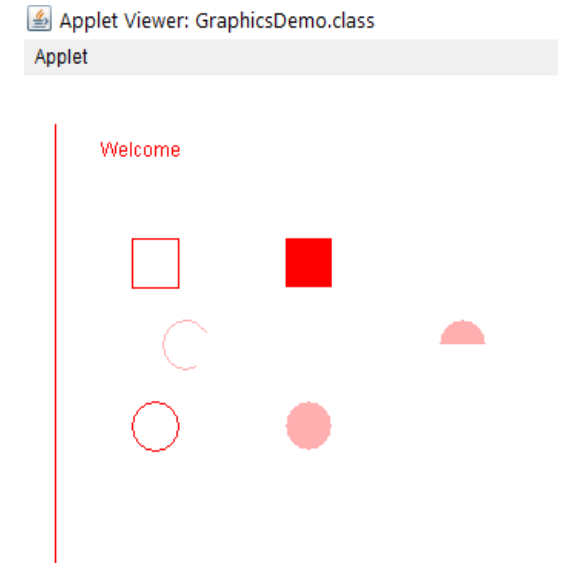
Run The Program by using HTML File

Create a new text File(in notepad)

```
<html>
<body>
<applet    code="    GraphicsDemo.class"    width="300"
           height="300">
</applet>
</body>
</html>
```

Save File :- Test.html

Run :- appletviewer Test.html



Event :

- It is an object that describes a state change in a source

Event Source:

- A source is an object that generate event.

Event Delegation Model:

- A source generate an event and send it to one or more listeners.
- The listener simply waits until it receives an event.
- Once it received event the listener process the event and then return.

Steps:

1. Create component object(eg. Button checkbox)
2. Add the component on platform (applet, frame, panel)
3. Register the respective listener
4. Implement Action performed() method

List of Listeners

EVENTS	SOURCE	LISTENERS
Action Event	Button, List, MenuItem, Text field	ActionListener
Component Event	Component	Component Listener
Focus Event	Component	FocusListener
Item Event	Checkbox, CheckboxMen ultem, Choice, List	ItemListener
Key Event	when input is received from keyboard	KeyListener
Text Event	Text Component	TextListener
Window Event	Window	WindowListener
Mouse Event	Mouse related event	MouseListener

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="ButtonDemo" width=900 height=500 >
  </applet>
*/
public class ButtonDemo extends Applet implements ActionListener
{
    String msg = "";
    Button submit;
    public void init() {
        submit = new Button("::Show Message::");
        add(submit);
        submit.addActionListener(this);
    }
}
```

Example ActionEvent

```
public void actionPerformed(ActionEvent ae) {  
    msg = "Welcome to CSE-III Semester in NIET Gr. Noida";  
    repaint();  
}  
public void paint(Graphics g) {  
    g.drawString(msg, 350, 100);  
}  
}
```


- it is used to arrange components in a particular manner.
- It facilitates us to control the positioning and size of the components in GUI forms.

classes that represent the layout managers:

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout

Java BorderLayout

- It is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window.
- **public static final int NORTH**
- **public static final int SOUTH**
- **public static final int EAST**
- **public static final int WEST**
- **public static final int CENTER**



Constructors of BorderLayout class:-

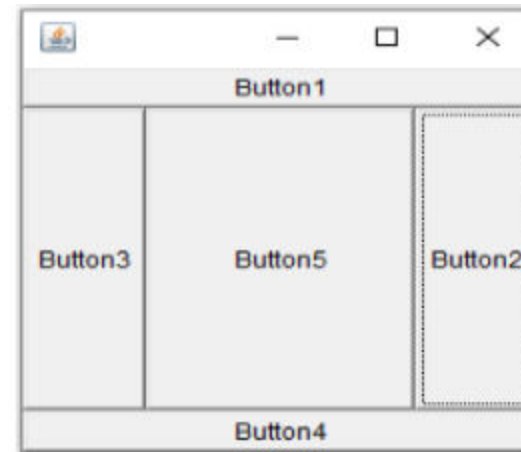
- **BorderLayout():** no gaps between the components.
- **BorderLayout(int hgap, int vgap):** gives horizontal and vertical gaps between the components.

Example to demonstrate Border Layout in Java

```
import java.awt.*;  
public class BorderLayoutDemo  
{  
    public static void main (String[]args)  
    {  
        Frame f1 = new Frame ();  
        f1.setSize (250, 250);  
        Button b1 = new Button ("Button1");  
        Button b2 = new Button ("Button2");  
        Button b3 = new Button ("Button3");  
        Button b4 = new Button ("Button4");  
        Button b5 = new Button ("Button5");  
        f1.add (b1, BorderLayout.NORTH);  
        f1.add (b2, BorderLayout.EAST);  
        f1.add (b3, BorderLayout.WEST);
```

```
        f1.add (b4, BorderLayout.SOUTH);  
        f1.add (b5);  
        f1.setVisible (true);  
    }  
}
```

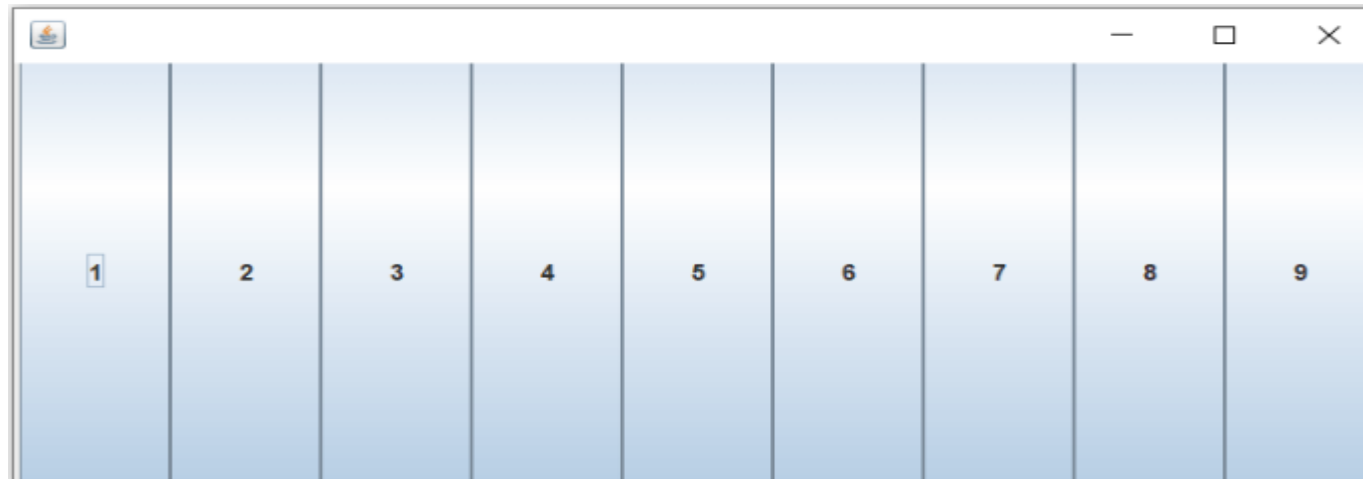
Output



it is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

- **GridLayout():** one column per component in a row.
- **GridLayout(int rows, int columns):** given rows and columns but no gaps between the components.
- **GridLayout(int rows, int columns, int hgap, int vgap):** given rows and columns along with given horizontal and vertical gaps.

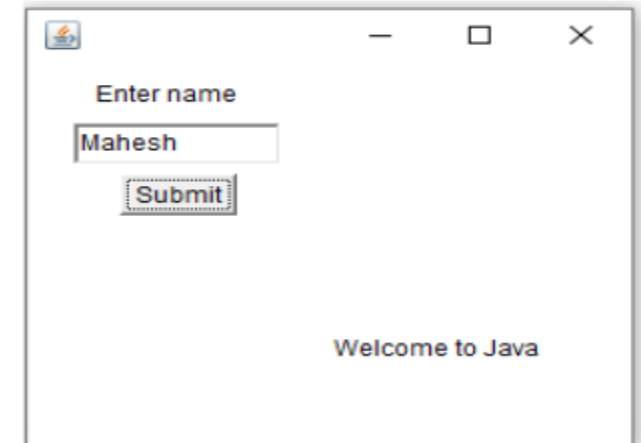


Example to demonstrate Grid Layout in Java

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutDemo
{
    public static void main (String[]args)
    {
        Frame f1 = new Frame ();
        f1.setSize (250, 250);
        GridLayout ob = new GridLayout (2, 2);
        f1.setLayout (ob);
        Panel p1 = new Panel ();
        Label l1 = new Label ("Enter name");
        TextField tf = new TextField (10);
        Button b1 = new Button ("Submit");
```

```
        p1.add (l1);
        p1.add (tf);
        p1.add (b1);
        f1.add (p1);
        Panel p2 = new Panel ();
        f1.add (p2);
        Panel p3 = new Panel ();
        f1.add (p3);
        Label l2 = new Label ("Welcome to Java");
        f1.add (l2);
        f1.setVisible (true);
    }
}
```

Output



- it is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.
- Fields of FlowLayout class
- **public static final int LEFT**
- **public static final int RIGHT**
- **public static final int CENTER**
- **public static final int LEADING**
- **public static final int TRAILING**



Constructors of FlowLayout class

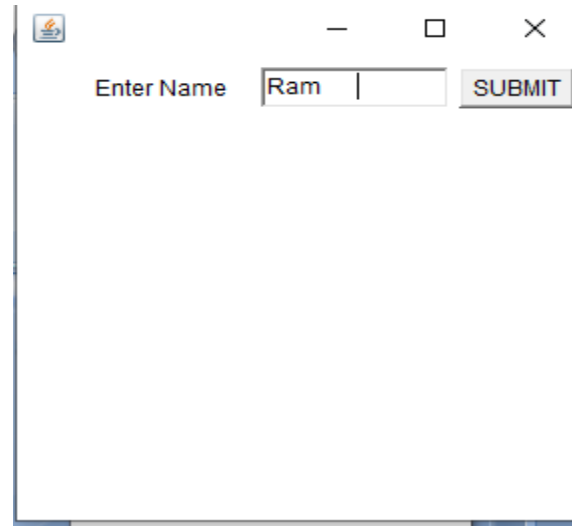
- **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example to demonstrate Flow Layout in Java

```
import java.awt.*;  
//import javax.swing.*;  
public class FlowLayoutDemo  
{  
    Frame f;  
    FlowLayoutDemo ()  
    {  
        f = new Frame ();  
        Label l1 = new Label ("Enter Name");  
        TextField tf1 = new TextField (10);  
        Button b1 = new Button ("SUBMIT");  
        f.add (l1);  
        f.add (tf1);  
        f.add (b1);  
        f.setLayout (new FlowLayout (FlowLayout.RIGHT));  
    }  
}
```

```
//setting flow layout of right alignment  
f.setSize (300, 300);  
f.setVisible (true);  
}  
public static void main (String[]args)  
{  
    new FlowLayoutDemo ();  
}  
}
```

Output



- The **Java BoxLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants.
- Fields of BoxLayout Class
- **public static final int X_AXIS:** Alignment of the components are horizontal from left to right.
- **public static final int Y_AXIS:** Alignment of the components are vertical from top to bottom.
- **public static final int LINE_AXIS:** - Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container.
- **public static final int PAGE_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container.

Constructor of BoxLayout class

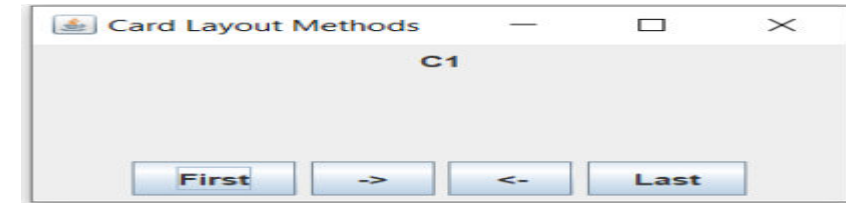
- **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.



it manages the components in such that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout Class

- **CardLayout():** zero horizontal and vertical gap.
- **CardLayout(int hgap, int vgap)** with the given horizontal and vertical gap.

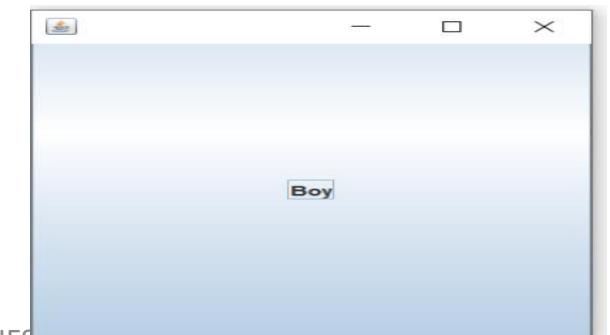


Commonly Used Methods of CardLayout Class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.



When the button named apple is clicked, we get

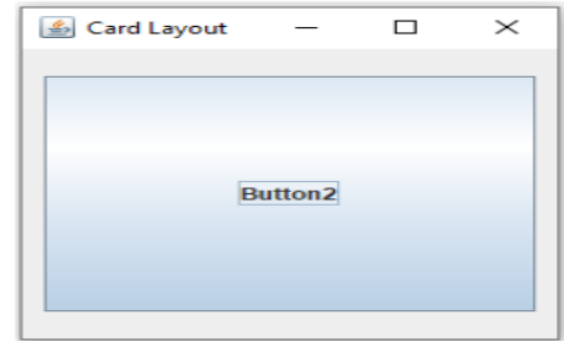


Example to demonstrate Card Layout in Java

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

public class CardLayoutDemo1 extends JFrame
implements ActionListener
{
    JButton b1, b2, b3, b4, b5;
    CardLayout cl;
    Container c;
    CardLayoutDemo1 ()
    {
        b1 = new JButton ("Button1");
        b2 = new JButton ("Button2");
        b3 = new JButton ("Button3");
        b4 = new JButton ("Button4");
        b5 = new JButton ("Button5");
```

```
        c = this.getContentPane ();
        cl = new CardLayout (10, 20);
        c.setLayout (cl);
        c.add ("Card1", b1);
        c.add ("Card2", b2);
        c.add ("Card3", b3);
        b1.addActionListener (this);
        b2.addActionListener (this);
        b3.addActionListener (this);
        setVisible (true);
        setSize (400, 400);
        setTitle ("Card Layout");
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    } public void actionPerformed (ActionEvent ae)
    {
        cl.next (c);
    } public static void main (String[] args)
    {
        new CardLayoutDemo ();
    }
}
```



GridBag Layout

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area.



Example:- 1

```
import java.awt.*;

public class Example4 extends java.applet.Applet
{
    public void init()
    {
        Panel p;
        setLayout(new BorderLayout());
        p = new Panel();
        p.add(new TextArea());
        add("Center", p);
        p = new Panel();
        p.add(new Button("One"));
        p.add(new Button("Two"));
        Choice c = new Choice();
```

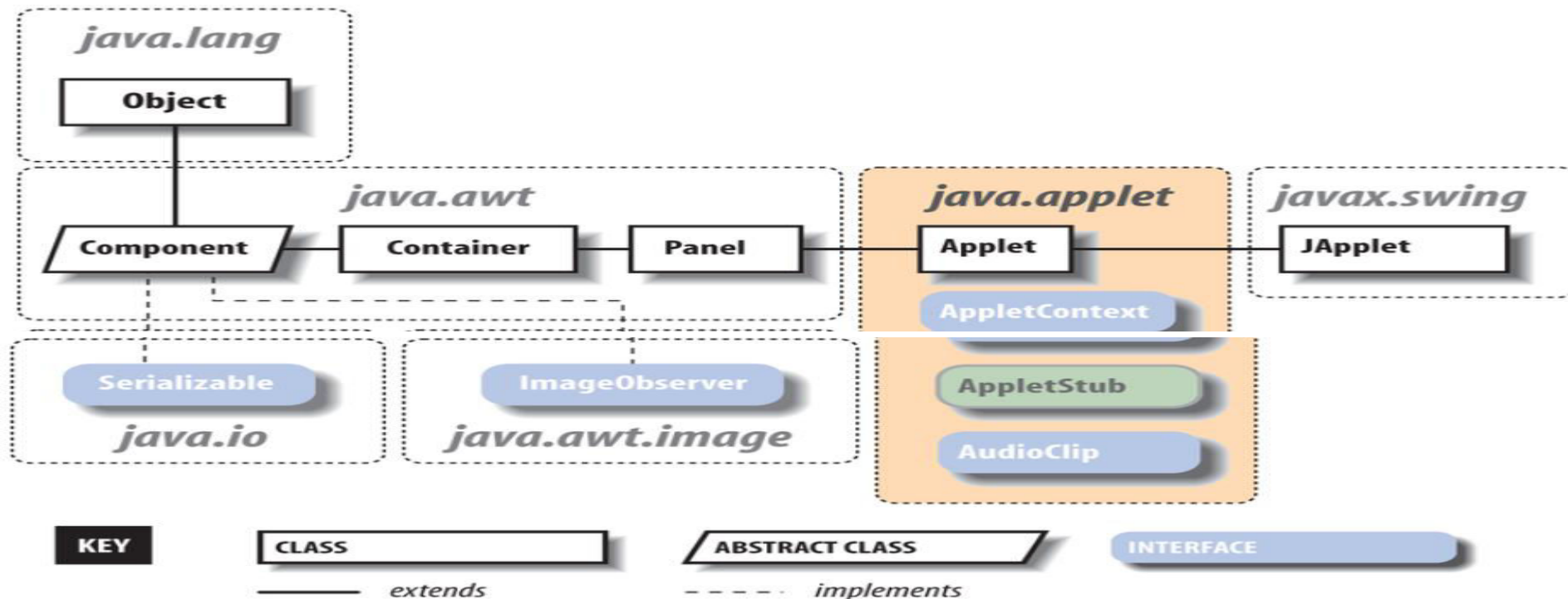
```
        c.addItem("one");
        c.addItem("two");
        c.addItem("three");
        p.add(c);
        add("North", p);
    }

    public static void main(String [] args)
    {
        Frame f = new Frame("Example 4");
        Example4 ex = new Example4();
        ex.init();
        f.add("Center", ex);
        f.pack();
        f.show();
    }
}
```

Output



- JApplet is a simple extension of `java.applet.Applet` to use when creating Swing programs designed to be used in a web browser (or *appletviewer*). As a direct subclass of `Applet`, JApplet is used in much the same way, with the `init()`, `start()`, and `stop()` methods still playing critical roles.



A-frame can be created in two ways

#1) By using the Frame class object

- Here, we create a Frame class object by instantiating the Frame class.

```
import java.awt.*;
```

```
class FrameButton{
```

```
    FrameButton (){
```

```
        Frame f=new Frame();
```

```
        Button b=new Button("CLICK_ME");
```

```
        b.setBounds(30,50,80,30);
```

```
        f.add(b);
```

```
        f.setSize(300,300);
```

```
        f.setLayout(null);
```

```
        f.setVisible(true);
```

```
    }
```

```
public static void main(String args[]){  
    FrameButton f=new FrameButton ();  
}  
}
```

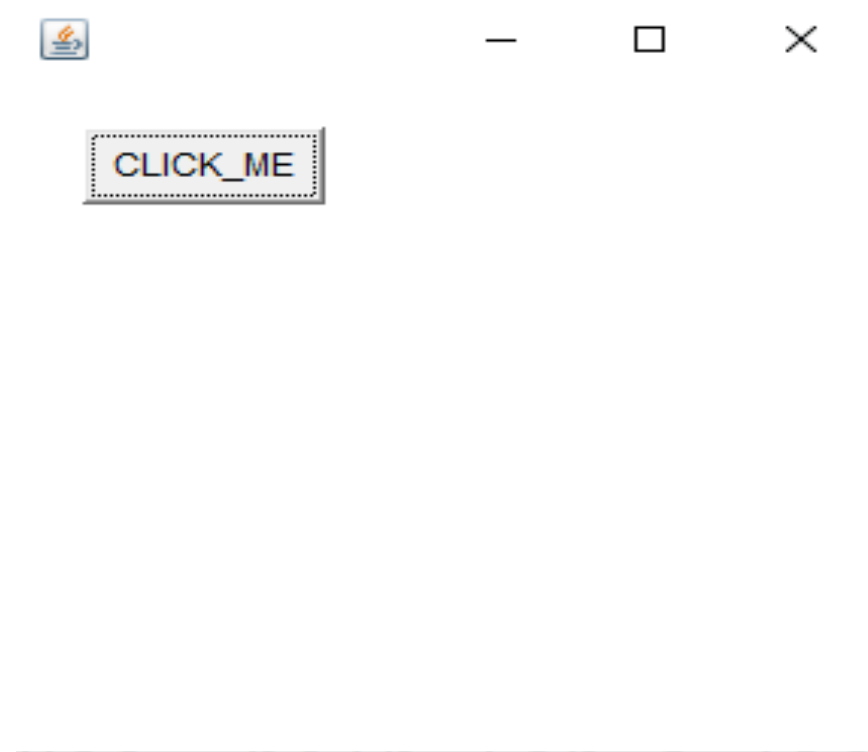
Save File :- FramButton.java

Output



#2) By Extending the Frame class

```
import java.awt.*;  
class AWTButton extends Frame{  
    AWTButton (){  
        Button b=new Button("AWTButton");  
        b.setBounds(30,100,80,30);// setting button  
position  
        add(b);//adding button into frame  
        setSize(300,300);//frame size 300 width and  
300 height  
        setLayout(null);//no layout manager  
        setVisible(true);//now frame will be visible,  
by default not visible  
    }  
    public static void main(String args[]){  
        AWTButton f=new AWTButton ();  
    }  
}
```



working with generics in java

Generics means **parameterized types**

The idea is to allow type (Integer, String, ... etc., and user-defined types) to be a parameter to methods, classes, and interfaces.

Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Generics Class

```
class MyGen<T>{  
    T obj;  
  
    void add(T obj)  
    { this.obj=obj;  
    }  
  
    T get()  
    { return obj;  
    }  
}
```

```
class TestGenerics3{  
    public static void main(String args[]){  
        MyGen<Integer> m=new MyGen<Integer>();  
        m.add(20);  
        System.out.println(m.get());  
  
        MyGen<String> m=new MyGen<String>();  
        m.add("Rohan");  
        System.out.println(m.get());  
    }  
}
```

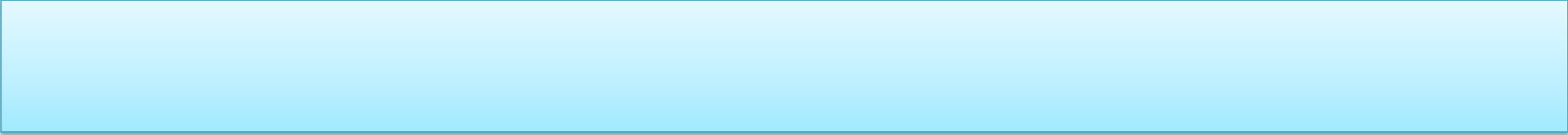
Output
20
Rohan

Generics Method

```
class MyGeneric {  
    // create a generics method  
    public <T> void gMethod(T data) {  
        System.out.println("Generics Method:");  
        System.out.println("Values: " + data);  
    }  
}
```

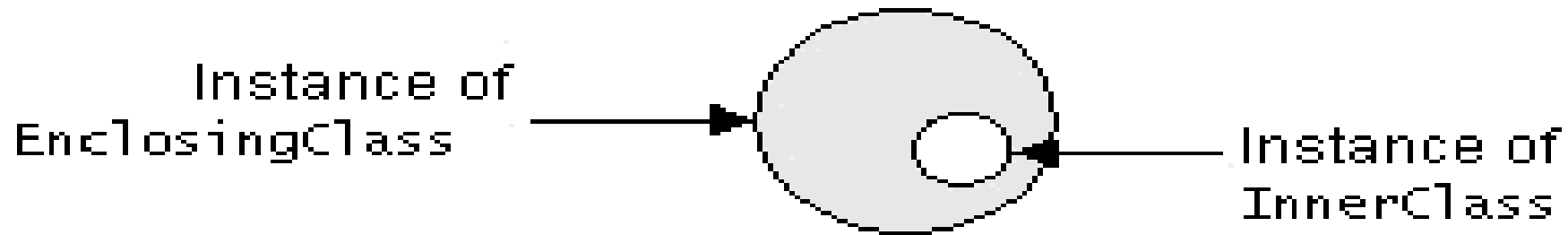
Output:
Generics Method:
Values: Hi!
Welcome
Generics Method:
Values: 25

```
class Test {  
    public static void main(String[] args) {  
        MyGeneric obj= new MyGeneric();  
        // for String data  
        obj.<String>gMethod("Hi! Welcome");  
        // for integer data  
        obj.<Integer>gMethod(25);  
    }  
}
```

- 
- Collections.
 - Using method references
 - Using Wrapper classes and Using Lists
 - Sets, Maps and Queues

Nested classes [CO5]

- **nested class:** A class defined inside of another class.
- Usefulness:
 - Nested classes are hidden from other classes (encapsulated).
 - Nested objects can access/modify the fields of their outer object.
- Event listeners are often defined as nested classes inside a GUI.



Nested class syntax

```
// enclosing outer class  
public class name {
```

```
    ...
```

```
    // nested inner class  
    private class name {
```

```
        ...
```

```
    }
```

```
}
```

- Only the outer class can see the nested class or make objects of it.
- Each nested object is associated with the outer object that created it, so it can access/modify that outer object's methods/fields.
 - If necessary, can refer to outer object as **OuterClassName.this**

Static inner classes

```
// enclosing outer class
public class name {
    ...

    // non-nested static inner class
    public static class name {
        ...
    }
}
```

- Static inner classes are *not* associated with a particular outer object.
- They cannot see the fields of the enclosing class.
- *Usefulness*: Clients can refer to and instantiate static inner classes:
Outer.Inner name = new Outer.Inner(params);

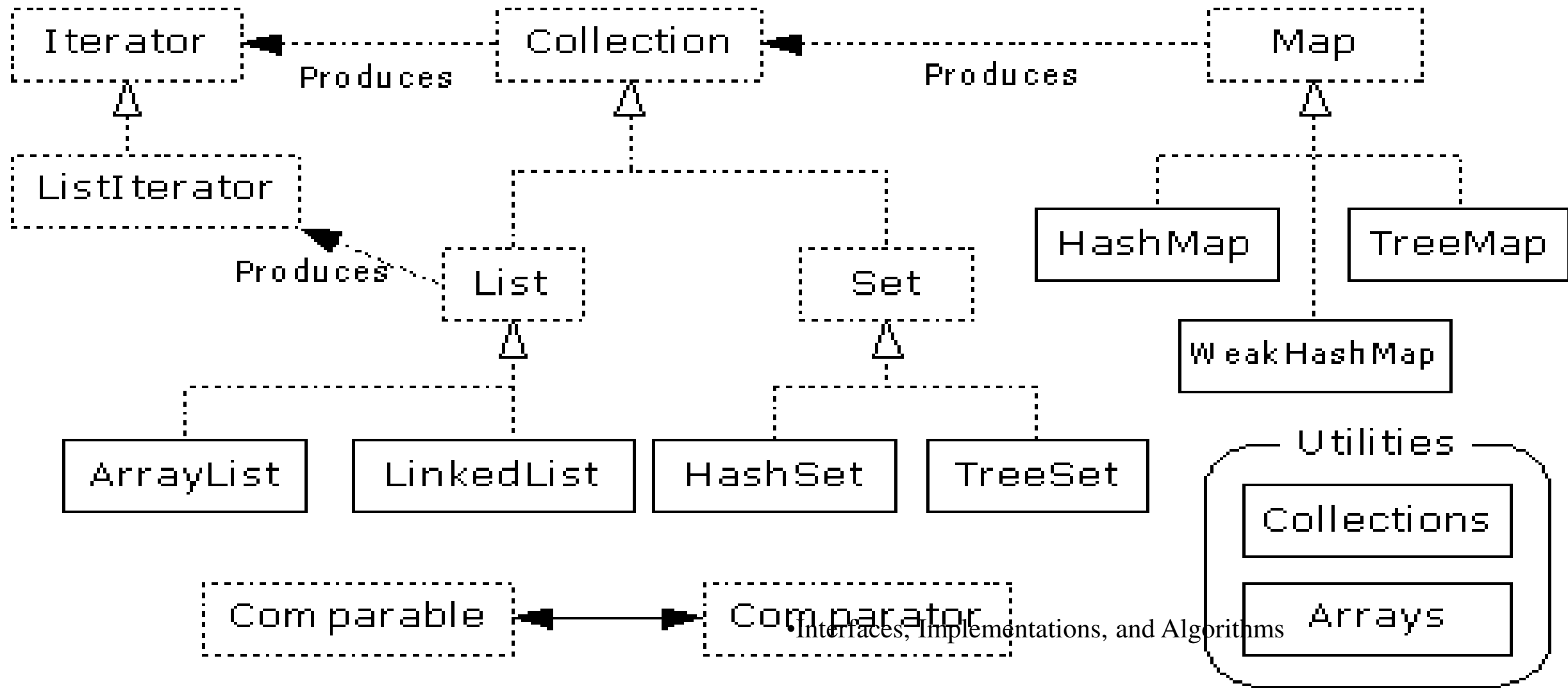
Java 2 Collections

- A collection is an object that groups multiple elements into a single unit
- Very useful
 - store, retrieve and manipulate data
 - transmit data from one method to another
 - data structures and methods written by hotshots in the field
 - Joshua Bloch, who also wrote the Collections tutorial

Collections Framework

- Unified architecture for representing and manipulating collections.
- A collections framework contains three things
 - Interfaces
 - Implementations
 - Algorithms

Collections Framework Diagram



Collection Interface

- Defines fundamental methods
 - **int size();**
 - **boolean isEmpty();**
 - **boolean contains(Object element);**
 - **boolean add(Object element); // Optional**
 - **boolean remove(Object element); // Optional**
 - **Iterator iterator();**
- These methods are enough to define the basic behavior of a collection
- Provides an Iterator to step through the elements in the Collection

Iterator Interface

- Defines three fundamental methods
 - `Object next()`
 - `boolean hasNext()`
 - `void remove()`
- These three methods provide access to the contents of the collection
- An Iterator knows position within collection
- Each call to `next()` “reads” an element from the collection
 - Then you can use it or remove it

Iterator Position

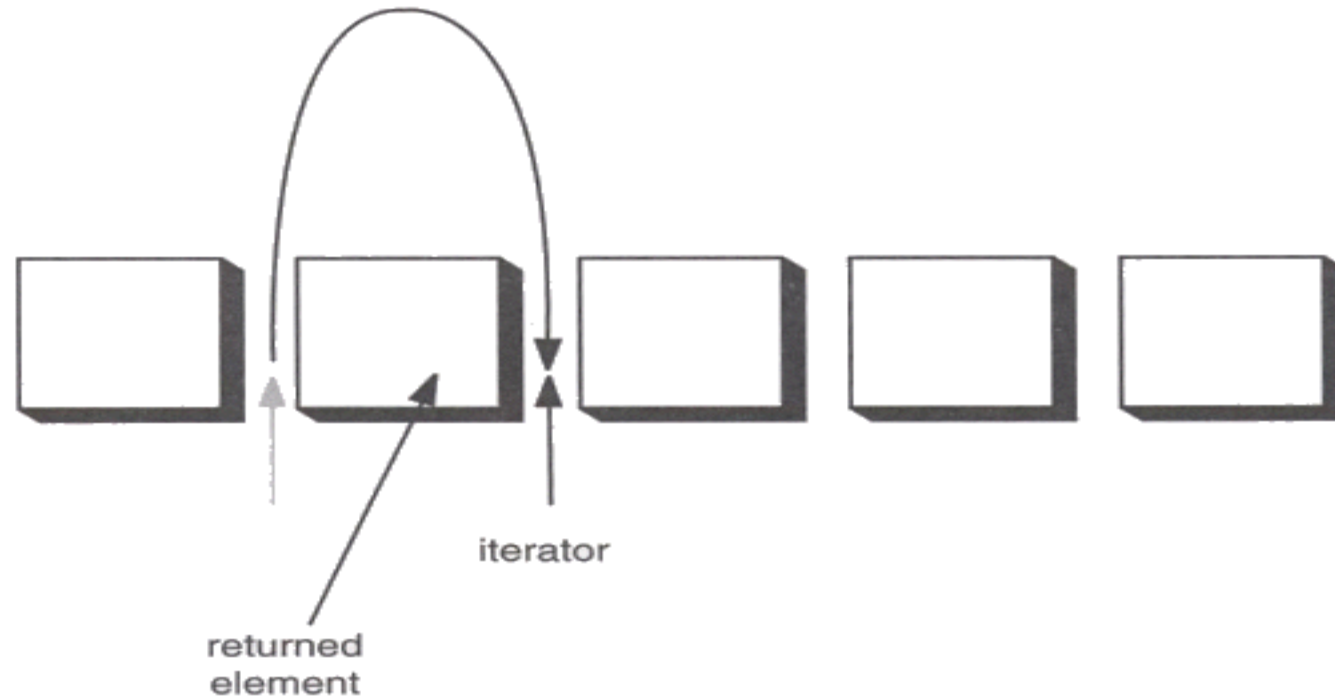
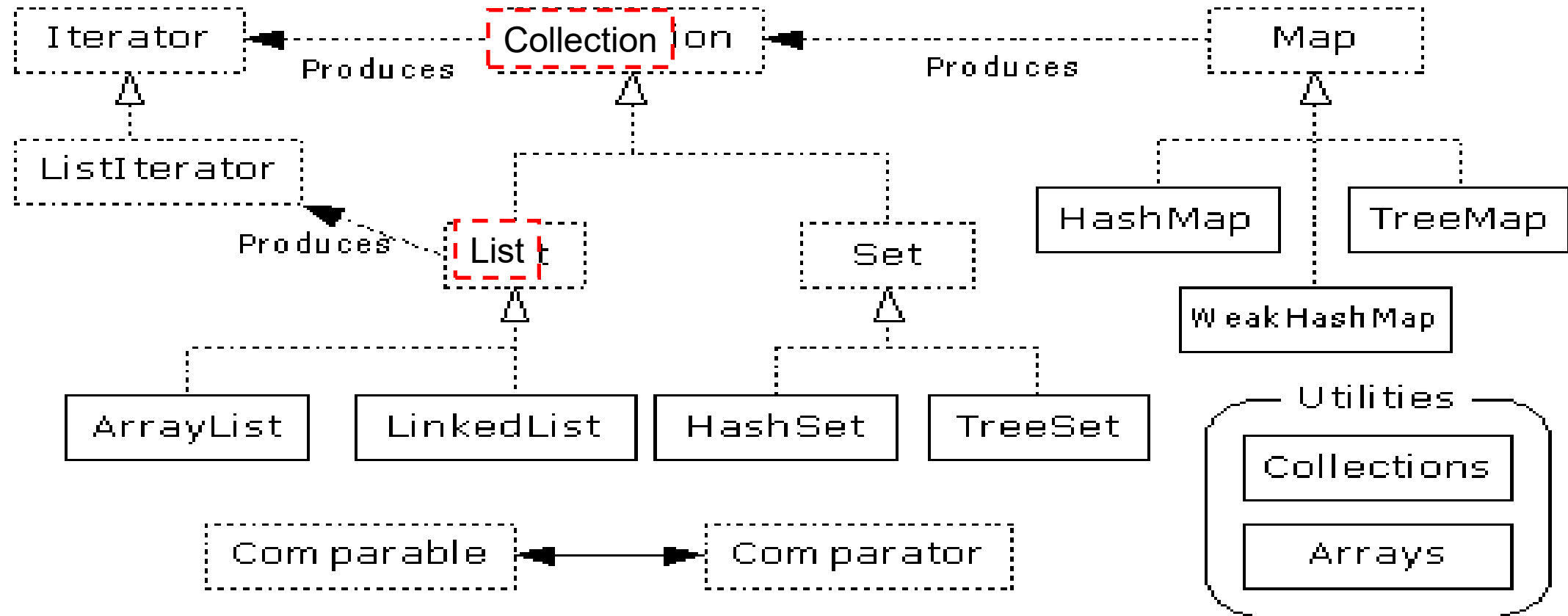


Figure 2-3: Advancing an iterator

Example - SimpleCollection

```
public class SimpleCollection {
    public static void main(String[] args) {
        Collection c;
        c = new ArrayList();
        System.out.println(c.getClass().getName());
        for (int i=1; i <= 10; i++) {
            c.add(i + " * " + i + " = "+i*i);
        }
        Iterator iter = c.iterator();
        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```

List Interface Context



List Interface

- The List interface adds the notion of *order* to a collection
- The user of a list has control over where an element is added in the collection
- Lists typically allow *duplicate* elements
- Provides a ListIterator to step through the elements in the list.

ListIterator Interface

- Extends the Iterator interface
- Defines three fundamental methods
 - **void add(Object o)** - before current position
 - **boolean hasPrevious()**
 - **Object previous()**
- The addition of these three methods defines the basic behavior of an ordered list
- A ListIterator knows position within list

Iterator Position - next(), previous()

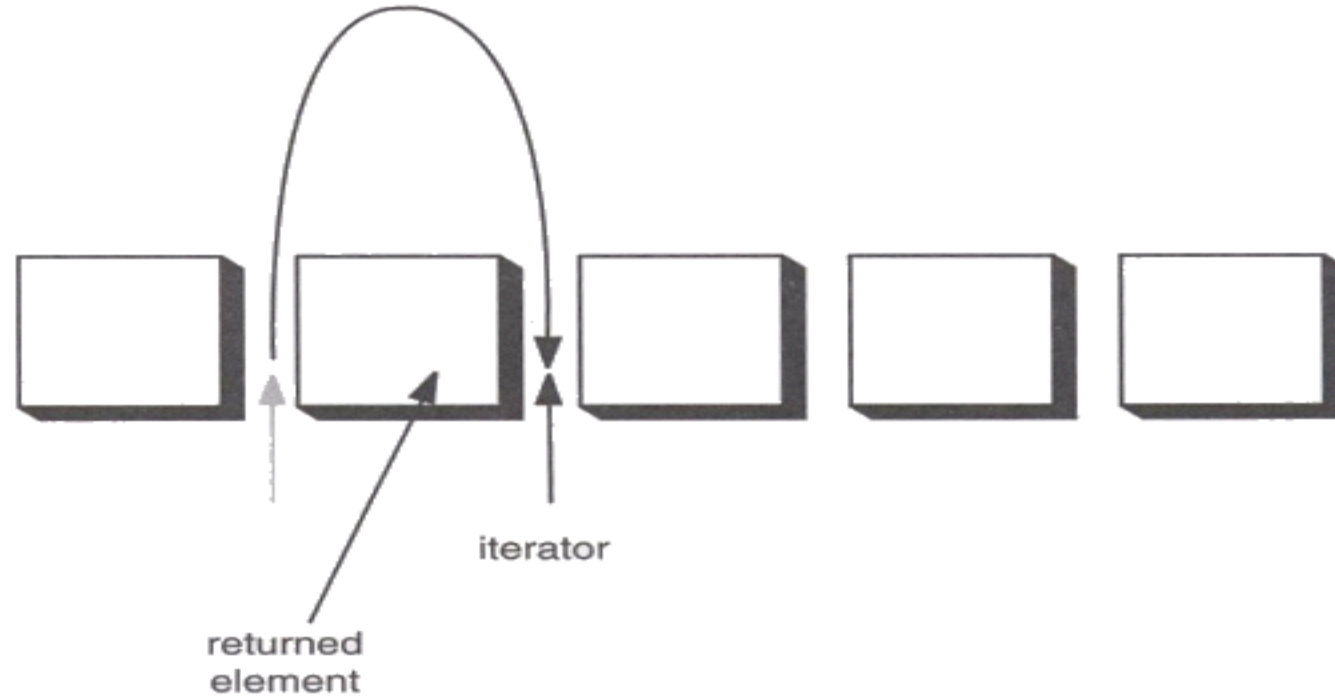
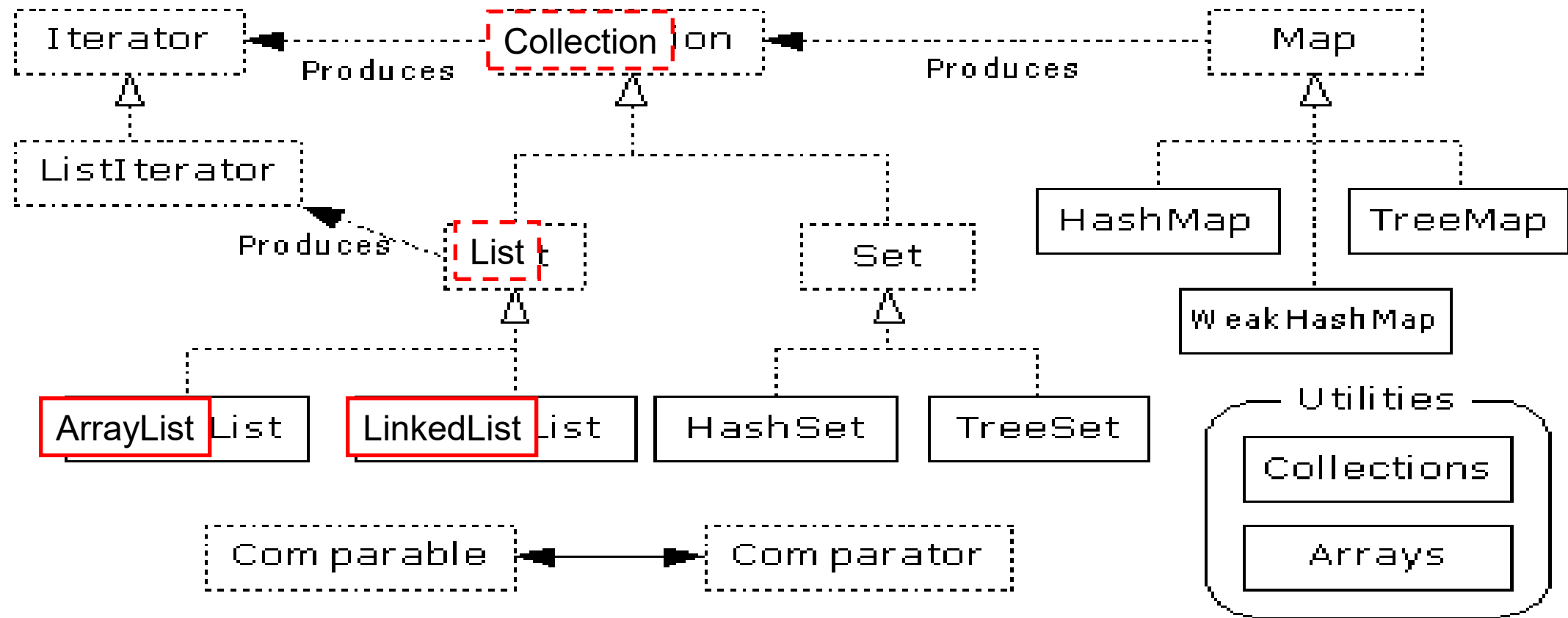


Figure 2-3: Advancing an iterator

ArrayList and LinkedList Context



List Implementations

- ArrayList
 - low cost random access
 - high cost insert and delete
 - array that resizes if need be
- LinkedList
 - sequential access
 - low cost insert and delete
 - high cost random access

ArrayList overview

- Constant time positional access (it's an array)
- One tuning parameter, the initial capacity

```
public ArrayList(int initialCapacity) {  
    super();  
    if (initialCapacity < 0)  
        throw new IllegalArgumentException(  
            "Illegal Capacity: "+initialCapacity);  
    this.elementData = new Object[initialCapacity];  
}
```

ArrayList methods

- The indexed get and set methods of the List interface are appropriate to use since ArrayLists are backed by an array
 - **Object get(int index)**
 - **Object set(int index, Object element)**
- Indexed add and remove are provided, but can be costly if used frequently
 - **void add(int index, Object element)**
 - **Object remove(int index)**
- May want to resize in one shot if adding many elements
 - **void ensureCapacity(int minCapacity)**

LinkedList overview

- Stores each element in a node
- Each node stores a link to the next and previous nodes
- Insertion and removal are inexpensive
 - just update the links in the surrounding nodes
- Linear traversal is inexpensive
- Random access is expensive
 - Start from beginning or end and traverse each node while counting

LinkedList entries

```
private static class Entry {
    Object element;
    Entry next;
    Entry previous;

    Entry(Object element, Entry next, Entry previous) {
        this.element = element;
        this.next = next;
        this.previous = previous;
    }
}

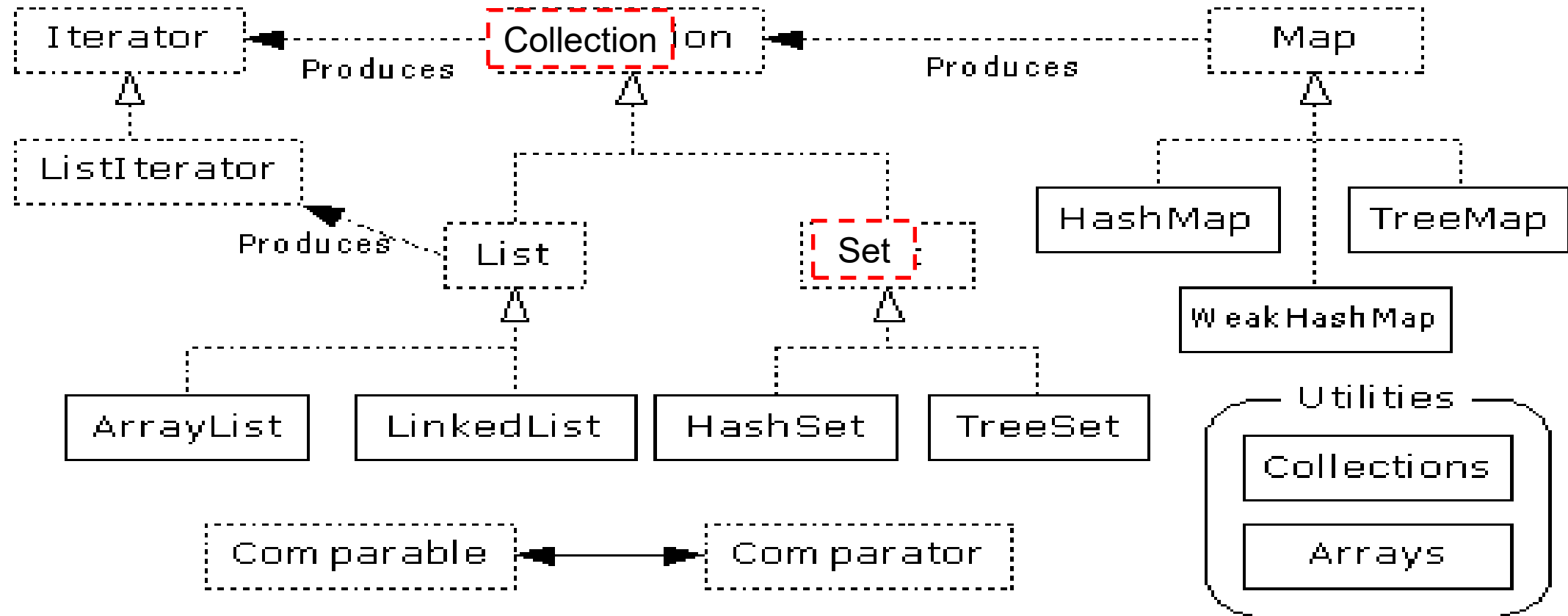
private Entry header = new Entry(null, null, null);

public LinkedList() {
    header.next = header.previous = header;
}
```

LinkedList methods

- The list is sequential, so access it that way
 - **ListIterator listIterator()**
- ListIterator knows about position
 - use **add()** from ListIterator to add at a position
 - use **remove()** from ListIterator to remove at a position
- LinkedList knows a few things too
 - **void addFirst(Object o), void addLast(Object o)**
 - **Object getFirst(), Object getLast()**
 - **Object removeFirst(), Object removeLast()**

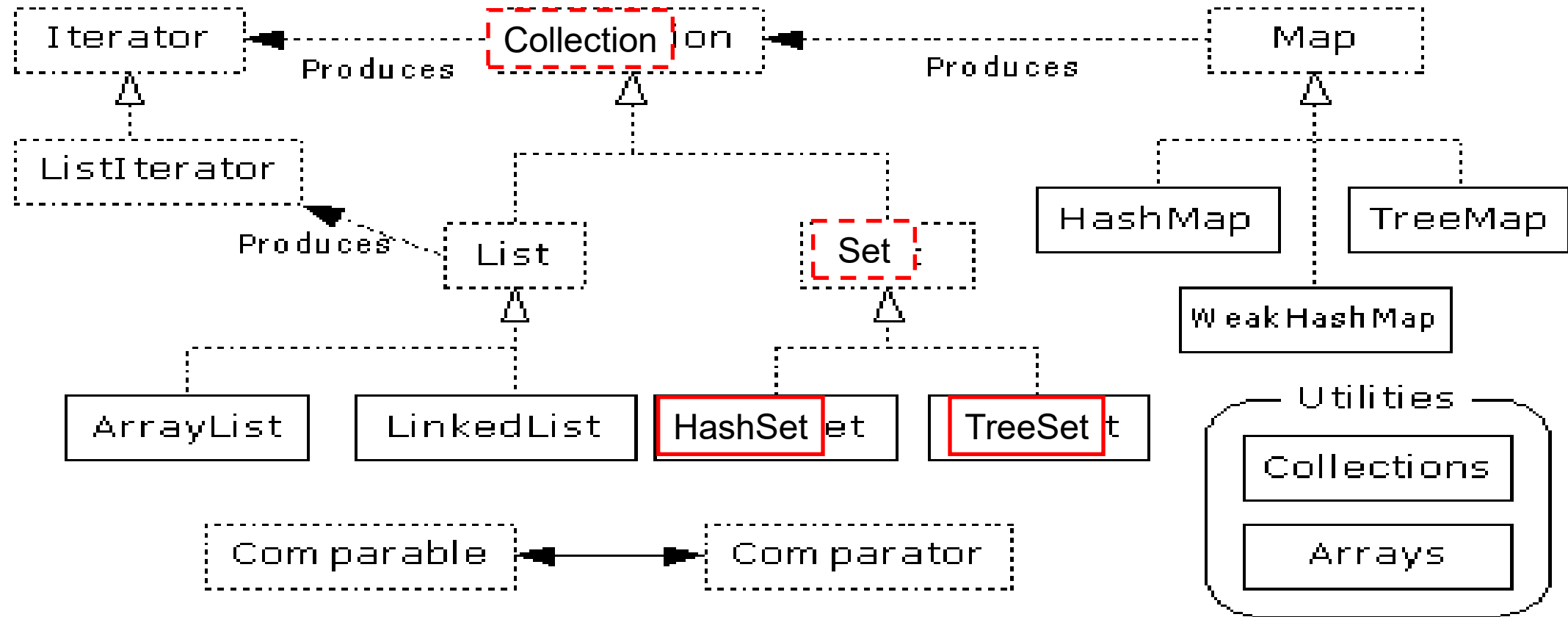
Set Interface Context



Set Interface

- Same methods as Collection
 - different contract - no duplicate entries
- Defines two fundamental methods
 - `boolean add(Object o)` - reject duplicates
 - `Iterator iterator()`
- Provides an Iterator to step through the elements in the Set
 - No guaranteed order in the basic Set interface
 - There is a SortedSet interface that extends Set

HashSet and TreeSet Context [CO5]



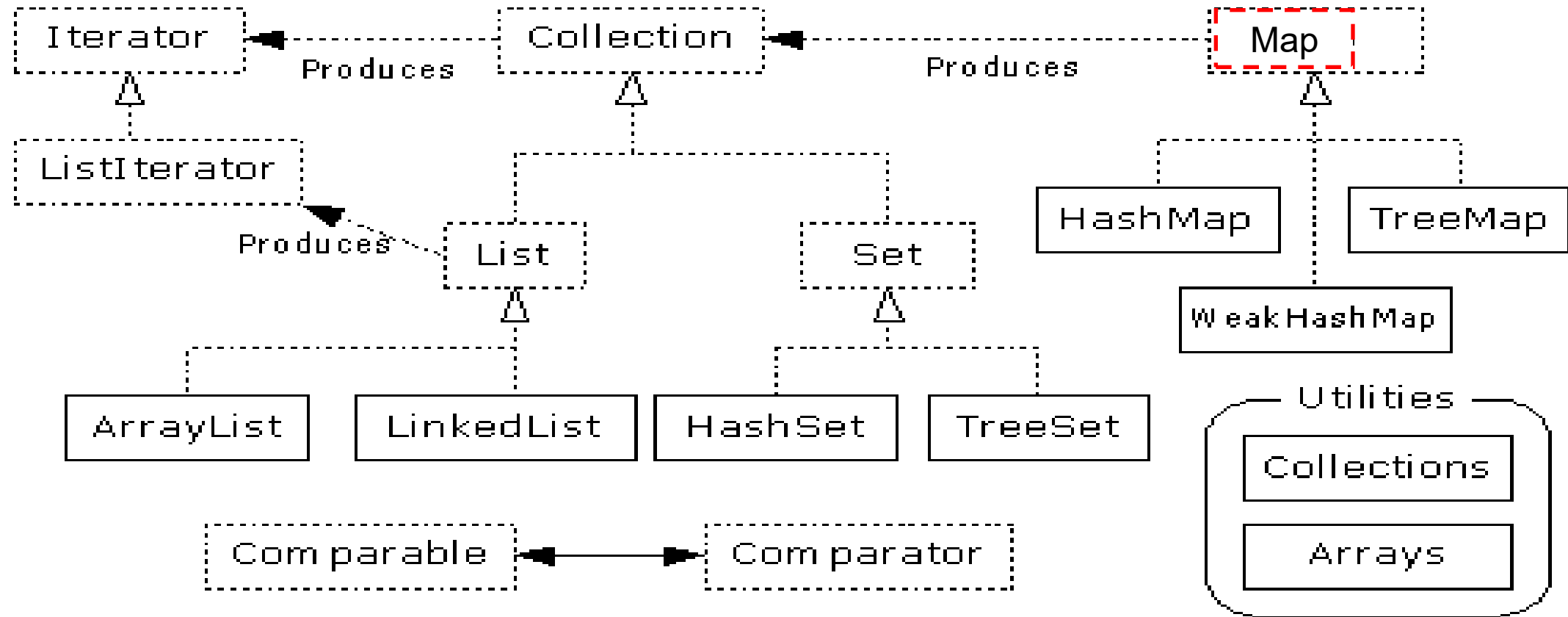
HashSet

- Find and add elements very quickly
 - uses hashing implementation in HashMap
- Hashing uses an array of linked lists
 - The **hashCode ()** is used to index into the array
 - Then **equals ()** is used to determine if element is in the (short) list of elements at that index
- No order imposed on elements
- The **hashCode ()** method and the **equals ()** method must be compatible
 - if two objects are equal, they must have the same **hashCode ()** value

TreeSet

- Elements can be inserted in any order
- The TreeSet stores them in order
 - Red-Black Trees out of Cormen-Leiserson-Rivest
- An iterator always presents them in order
- Default order is defined by natural order
 - objects implement the Comparable interface
 - TreeSet uses **compareTo (Object o)** to sort
- Can use a different Comparator
 - provide Comparator to the TreeSet constructor

Map Interface Context



Map Interface [C05]

- Stores key/value pairs
- Maps from the key to the value
- Keys are unique
 - a single key only appears once in the Map
 - a key can map to only one value
- Values do not have to be unique

Map methods

Object put(Object key, Object value)

Object get(Object key)

Object remove(Object key)

boolean containsKey(Object key)

boolean containsValue(Object value)

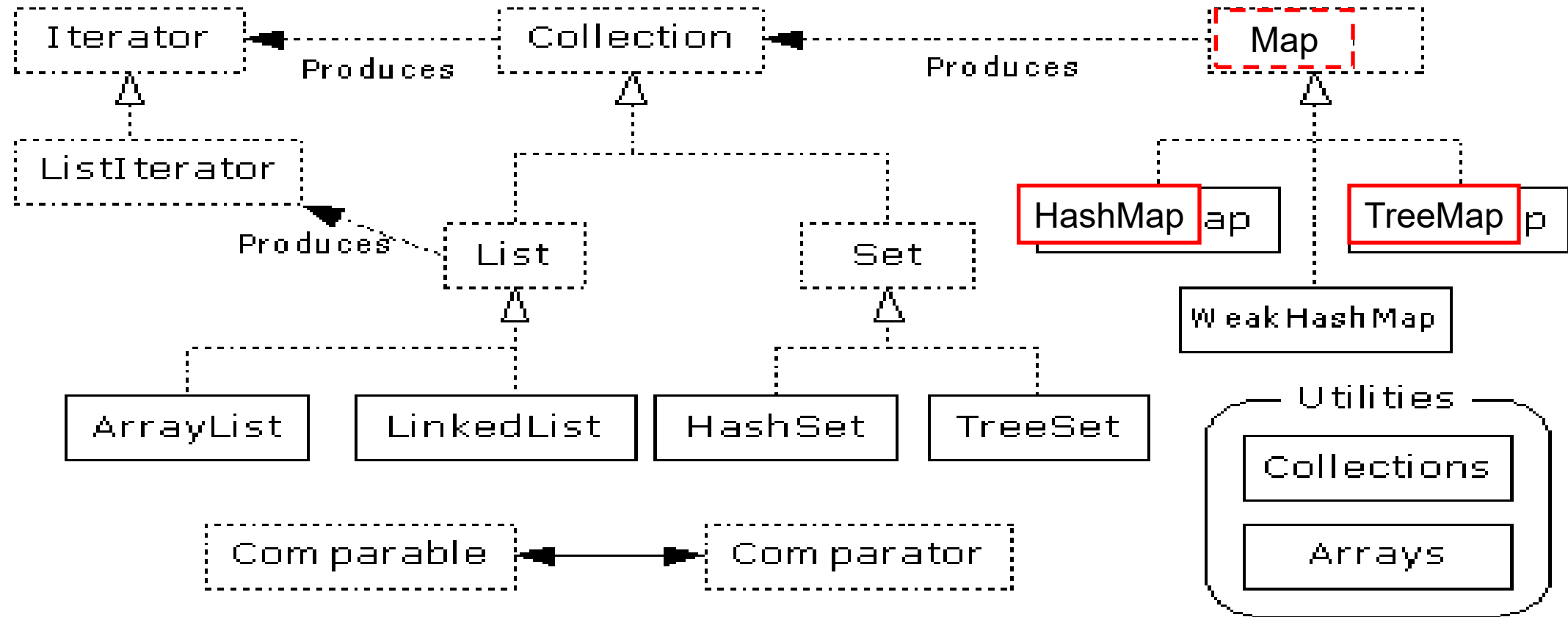
int size()

boolean isEmpty()

Map views

- A means of iterating over the keys and values in a Map
- **Set keySet()**
 - returns the Set of keys contained in the Map
- **Collection values()**
 - returns the Collection of values contained in the Map. This Collection is not a Set, as multiple keys can map to the same value.
- **Set entrySet()**
 - returns the Set of key-value pairs contained in the Map. The Map interface provides a small nested interface called Map.Entry that is the type of the elements in this Set.

HashMap and TreeMap Context



HashMap and TreeMap

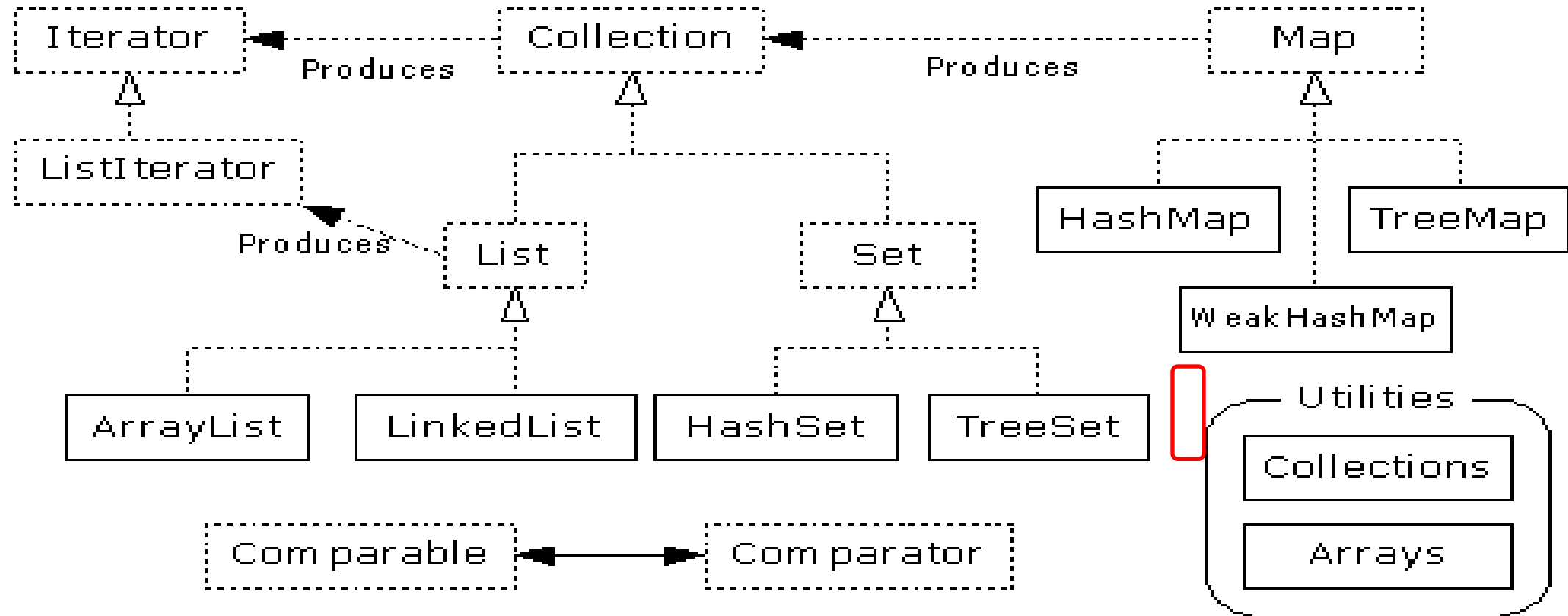
- HashMap
 - The keys are a set - unique, unordered
 - Fast
- TreeMap
 - The keys are a set - unique, ordered
 - Same options for ordering as a TreeSet
 - *Natural order (Comparable, compareTo(Object))*
 - *Special order (Comparator, compare(Object, Object))*

Bulk Operations

- In addition to the basic operations, a Collection may provide “bulk” operations

```
boolean containsAll(Collection c);  
boolean addAll(Collection c); // Optional  
boolean removeAll(Collection c); // Optional  
boolean retainAll(Collection c); // Optional  
void clear(); // Optional  
Object[] toArray();  
Object[] toArray(Object a[]);
```

Utilities Context



Utilities

- Using Wrapper classes and Using Lists
- Sets, Maps and Queues

Utilities

- The Collections class provides a number of static methods for fundamental algorithms
- Most operate on Lists, some on all Collections
 - Sort, Search, Shuffle
 - Reverse, fill, copy
 - Min, max
- Wrappers
 - synchronized Collections, Lists, Sets, etc
 - unmodifiable Collections, Lists, Sets, etc

More Legacy classes

- Vector
 - use ArrayList
- Stack
 - use LinkedList
- BitSet
 - use ArrayList of boolean, unless you can't stand the thought of the wasted space
- Properties
 - legacies are sometimes hard to walk away from ...
 - see next few pages

Properties class

- Located in java.util package
- Special case of Hashtable
 - Keys and values are Strings
 - Tables can be saved to/loaded from file