OBJECT ORIENTED TECHNIQUES USING JAVA (ACSE0302)

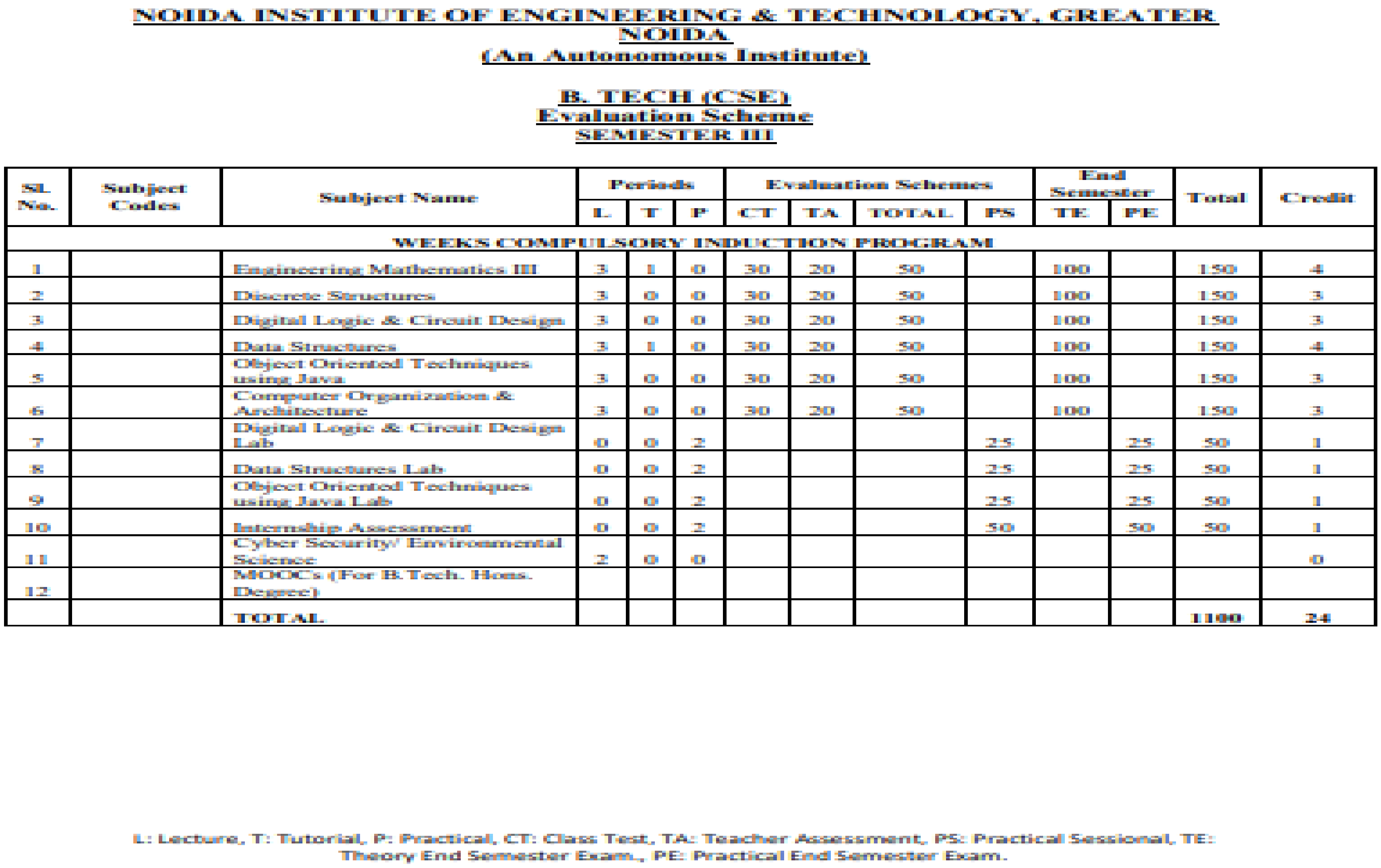
Unit: V

GUI Programming, Generics and Collections

B. TECH - 3rd Sem (CSE)

Divya Maheshwari
Assistant Professor
IT Department

Evaluation Scheme



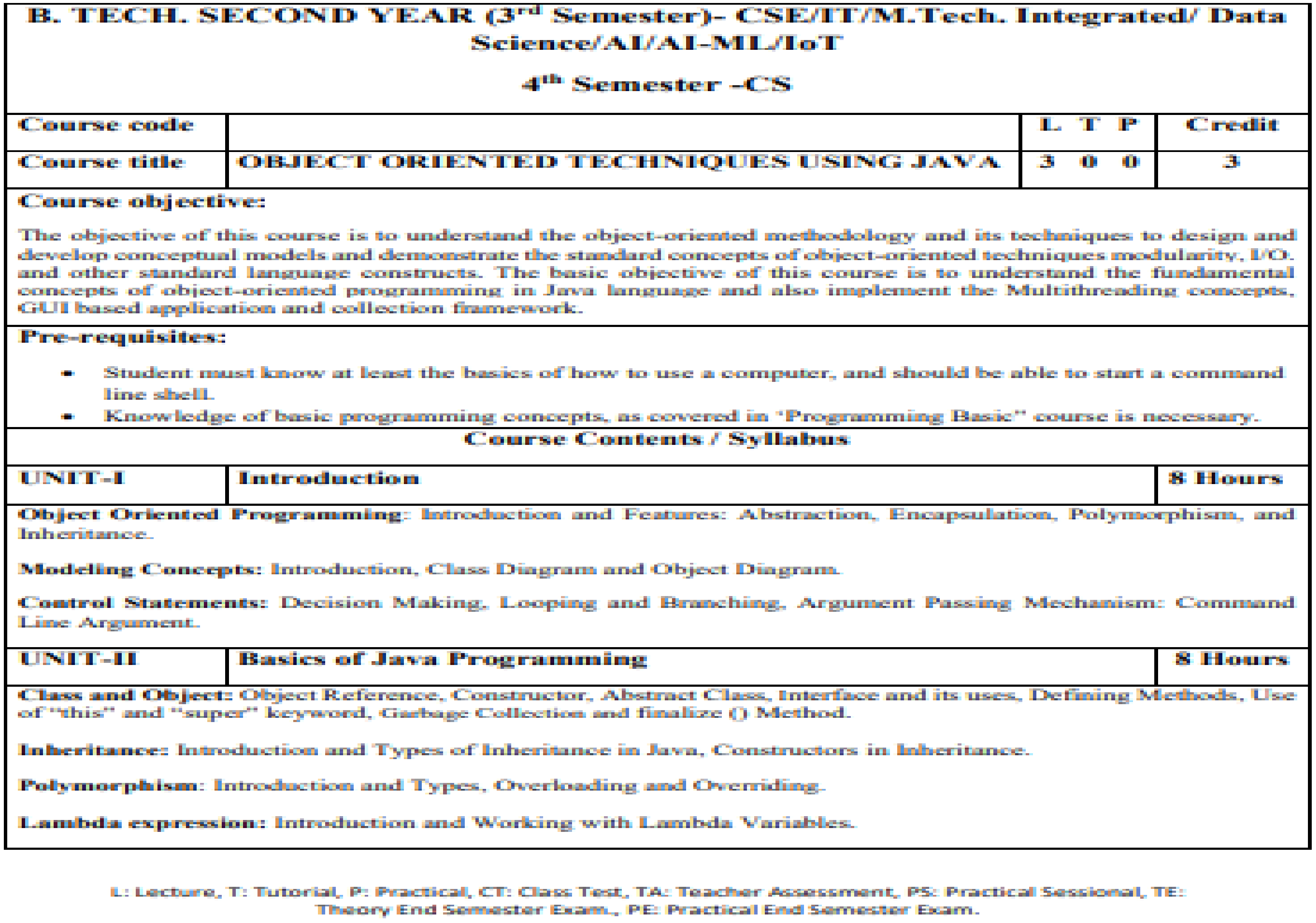
NOIDA INSTITUTE OF ENGINEERING & TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)

B. TECH (CSE)
Evaluation Scheme
SEMESTER III

Sl. No.	Subject Codes	Subject Name	Periods			Evaluation Schemes				End Semester		Total	Credit
			L	T	P	CT	TA	TOTAL	PS	TE	PE		
WEEKS COMPULSORY INDUCTION PROGRAM													
1		Engineering Mathematics III	3	1	0	30	20	50		100		150	4
2		Discrete Structures	3	0	0	30	20	50		100		150	3
3		Digital Logic & Circuit Design	3	0	0	30	20	50		100		150	3
4		Data Structures	3	1	0	30	20	50		100		150	4
5		Object Oriented Techniques using Java	3	0	0	30	20	50		100		150	3
6		Computer Organization & Architecture	3	0	0	30	20	50		100		150	3
7		Digital Logic & Circuit Design Lab	0	0	2				25		25	50	1
8		Data Structures Lab	0	0	2				25		25	50	1
9		Object Oriented Techniques using Java Lab	0	0	2				25		25	50	1
10		Internship Assessment	0	0	2				50		50	50	1
11		Cyber Security/ Environmental Science	2	0	0								0
12		MOOCs (For B.Tech. Hons. Degree)											
		TOTAL										1100	24

L: Lecture, T: Tutorial, P: Practical, CT: Class Test, TA: Teacher Assessment, PS: Practical Sessional, TE: Theory End Semester Exam., PE: Practical End Semester Exam.

Syllabus



B. TECH. SECOND YEAR (3rd Semester)- CSE/IT/M.Tech. Integrated/ Data Science/AI/AI-ML/IoT

4th Semester -CS

Course code		**L**	**T**	**P**	**Credit**
Course title	**OBJECT ORIENTED TECHNIQUES USING JAVA**	**3**	**0**	**0**	**3**

Course objective:

The objective of this course is to understand the object-oriented methodology and its techniques to design and develop conceptual models and demonstrate the standard concepts of object-oriented techniques modularity, I/O. and other standard language constructs. The basic objective of this course is to understand the fundamental concepts of object-oriented programming in Java language and also implement the Multithreading concepts, GUI based application and collection framework.

Pre-requisites:

- Student must know at least the basics of how to use a computer, and should be able to start a command line shell.
- Knowledge of basic programming concepts, as covered in "Programming Basic" course is necessary.

Course Contents / Syllabus

UNIT-I	**Introduction**	**8 Hours**

Object Oriented Programming: Introduction and Features: Abstraction, Encapsulation, Polymorphism, and Inheritance.

Modeling Concepts: Introduction, Class Diagram and Object Diagram.

Control Statements: Decision Making, Looping and Branching, Argument Passing Mechanism: Command Line Argument.

UNIT-II	**Basics of Java Programming**	**8 Hours**

Class and Object: Object Reference, Constructor, Abstract Class, Interface and its uses, Defining Methods, Use of "this" and "super" keyword, Garbage Collection and finalize () Method.

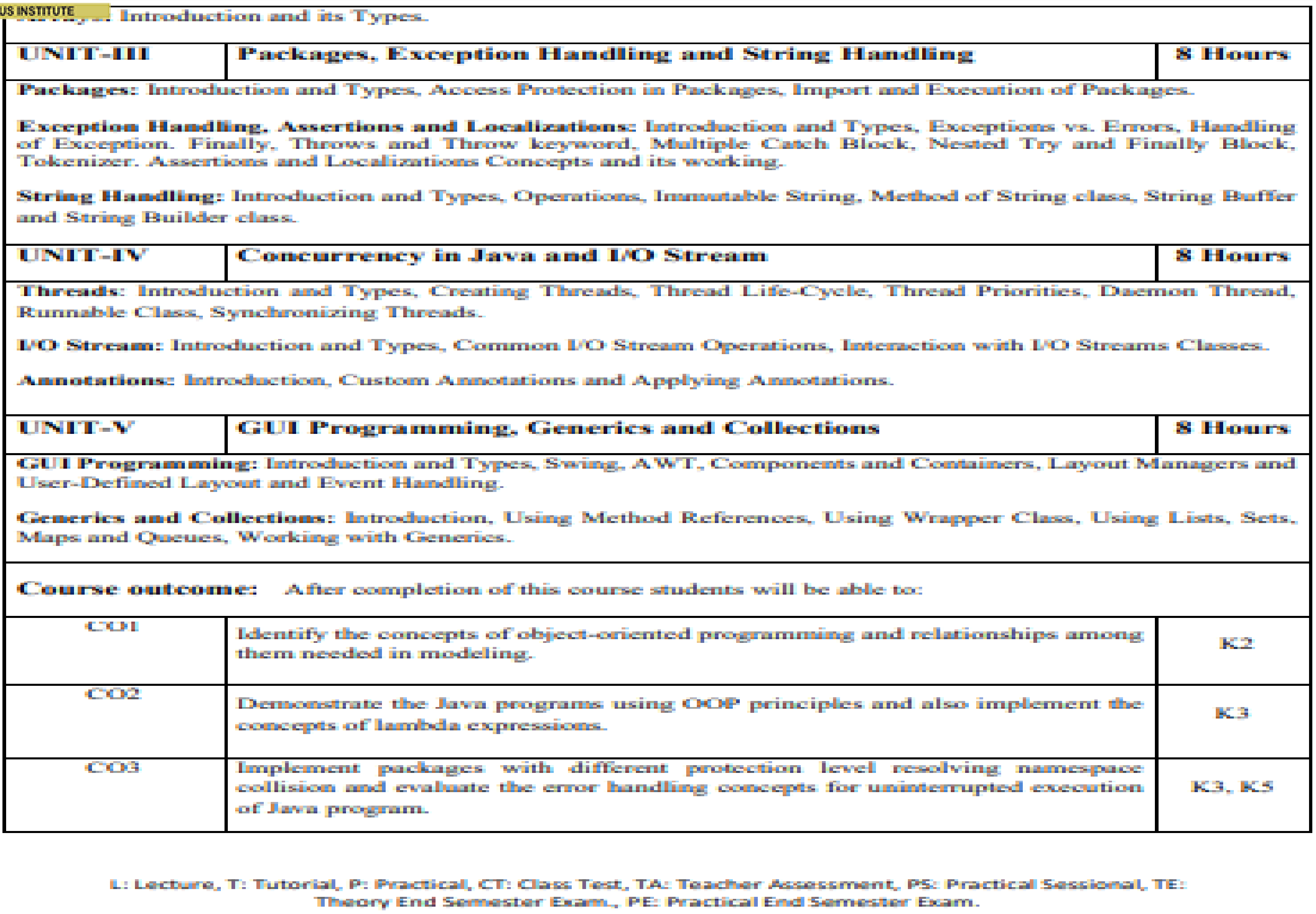
Inheritance: Introduction and Types of Inheritance in Java, Constructors in Inheritance.

Polymorphism: Introduction and Types, Overloading and Overriding.

Lambda expression: Introduction and Working with Lambda Variables.

L: Lecture, T: Tutorial, P: Practical, CT: Class Test, TA: Teacher Assessment, PS: Practical Sessional, TE: Theory End Semester Exam., PE: Practical End Semester Exam.

Syllabus



Introduction and its Types.		
UNIT-III	**Packages, Exception Handling and String Handling**	**8 Hours**
Packages: Introduction and Types, Access Protection in Packages, Import and Execution of Packages. **Exception Handling, Assertions and Localizations:** Introduction and Types, Exceptions vs. Errors, Handling of Exception. Finally, Throws and Throw keyword, Multiple Catch Block, Nested Try and Finally Block, Tokenizer. Assertions and Localizations Concepts and its working. **String Handling:** Introduction and Types, Operations, Immutable String, Method of String class, String Buffer and String Builder class.		
UNIT-IV	**Concurrency in Java and I/O Stream**	**8 Hours**
Threads: Introduction and Types, Creating Threads, Thread Life-Cycle, Thread Priorities, Daemon Thread, Runnable Class, Synchronizing Threads. **I/O Stream:** Introduction and Types, Common I/O Stream Operations, Interaction with I/O Streams Classes. **Annotations:** Introduction, Custom Annotations and Applying Annotations.		
UNIT-V	**GUI Programming, Generics and Collections**	**8 Hours**
GUI Programming: Introduction and Types, Swing, AWT, Components and Containers, Layout Managers and User-Defined Layout and Event Handling. **Generics and Collections:** Introduction, Using Method References, Using Wrapper Class, Using Lists, Sets, Maps and Queues, Working with Generics.		
Course outcome: After completion of this course students will be able to:		
CO1	Identify the concepts of object-oriented programming and relationships among them needed in modeling.	K2
CO2	Demonstrate the Java programs using OOP principles and also implement the concepts of lambda expressions.	K3
CO3	Implement packages with different protection level resolving namespace collision and evaluate the error handling concepts for uninterrupted execution of Java program.	K3, K5

L: Lecture, T: Tutorial, P: Practical, CT: Class Test, TA: Teacher Assessment, PS: Practical Sessional, TE: Theory End Semester Exam., PE: Practical End Semester Exam.

Branch wise Applications

- Desktop GUI Applications
- Mobile Applications
- Enterprise Applications
- Scientific Applications
- Web-based Applications
- Embedded Systems
- Big Data Technologies

Course Objective

- The objective of this course is to understand the object-oriented methodology and its techniques to design and develop conceptual models and demonstrate the standard concepts of object-oriented techniques modularity, I/O, and other standard language constructs.
- The basic objective of this course is to understand the fundamental concepts of object-oriented programming in Java language and implement the Multithreading concepts, GUI based application and collection framework.

Course Outcome

SEMESTER	COURSE	S.NO	DESCRIPTION	Bloom's Taxonomy Level
2nd Semester	Object Oriented Techniques with Java	CO1	Identify the concepts of object-oriented programming and relationships among them needed in modeling.	K2
		CO2	Demonstrate the Java programs using OOP principles and implement the concepts of lambda expressions.	K3
		CO3	Implement packages with different protection level resolving namespace collision and evaluate the error handling concepts for uninterrupted execution of Java program.	K3, K5
		CO4	Implement Concurrency control, I/O Streams and Annotations concepts by using Java program.	K3
		CO5	Design and develop the GUI based application, Generics and Collections in Java programming language to solve the real-world problem	K6

Program Outcome

1. **Engineering knowledge**.
2. **Problem analysis:**
3. **Design/development of solutions:**
4. **Conduct investigations of complex problems:**
5. **Modern tool usage:**.
6. **The engineer and society:**
7. **Environment and sustainability:**
8. **Ethics:**
9. **Individual and team work:**
10. **Communication**
11. **Project management and finance**
12. **Life-long learning**

CO-PO Mapping

Mapping of Course Outcomes and Program Outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO.1	3	3	3	2	1				2			3
CO.2	3	3	3	2	1				2			3
CO.3	3	3	3	2	1				2			3
CO.4	3	3	3	2	1				2			3
CO.5	**3**	**3**	**3**	**2**	**1**				**2**			**3**

3= High, 2=Medium, 1=Low

CO- PSO Mapping

	PSO1	PSO2	PSO3	PSO4
CO.1	2	3	2	1
CO.2	2	3	2	1
CO.3	2	3	2	1
CO.4	2	3	2	1
CO.5	**2**	**3**	**2**	**1**

3= High, 2=Medium, 1=Low

Program Specific Outcomes

On successful completion of graduation degree the Engineering graduates will be able to:

PSO1: The ability to identify, analyze real world problems and design their ethical solutions using artificial intelligence, robotics, virtual/augmented reality, data analytics, block chain technology, and cloud computing.

PSO2: The ability to design and develop the hardware sensor devices and related interfacing software systems for solving complex engineering problems.

PSO3: The ability to understand inter disciplinary computing techniques and to apply them in the design of advanced computing.

PSO4: The ability to conduct investigation of complex problem with the help of technical, managerial, leadership qualities, and modern engineering tools provided by industry sponsored laboratories.

Why learn GUIs?

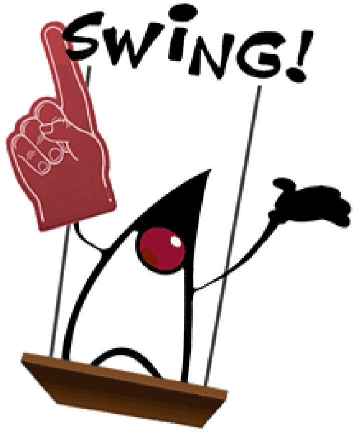
- Learn about *event-driven programming* techniques
- Practice learning and using a large, complex API
- A chance to see how it is designed and learn from it:
  - model-view separation
  - design patterns
  - refactoring vs. reimplementing an ailing API
- Because GUIs are neat!

- *Caution*: There is way more here than you can memorize.
  - Part of learning a large API is "letting go."
  - You won't memorize it all; you will look things up as you need them.
  - But you can learn the fundamental concepts and general ideas.

Java GUI History

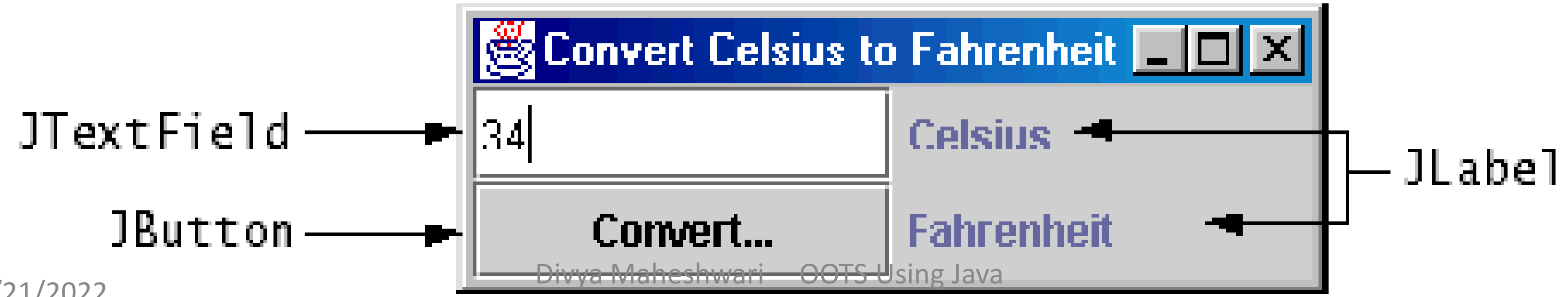
- **Abstract Windowing Toolkit** (**AWT**): Sun's initial effort to create a set of cross-platform GUI classes. *(JDK 1.0 - 1.1)*
  - Maps general Java code to each operating system's real GUI system.
  - *Problems:* Limited to lowest common denominator; clunky to use.

- **Swing**: A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction. *(JDK 1.2+)*
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
  - *Benefits:* Features; compatibility; OO design.
  - *Problem:* Both exist in Java now; easy to get them mixed up; still have to use both in various places.

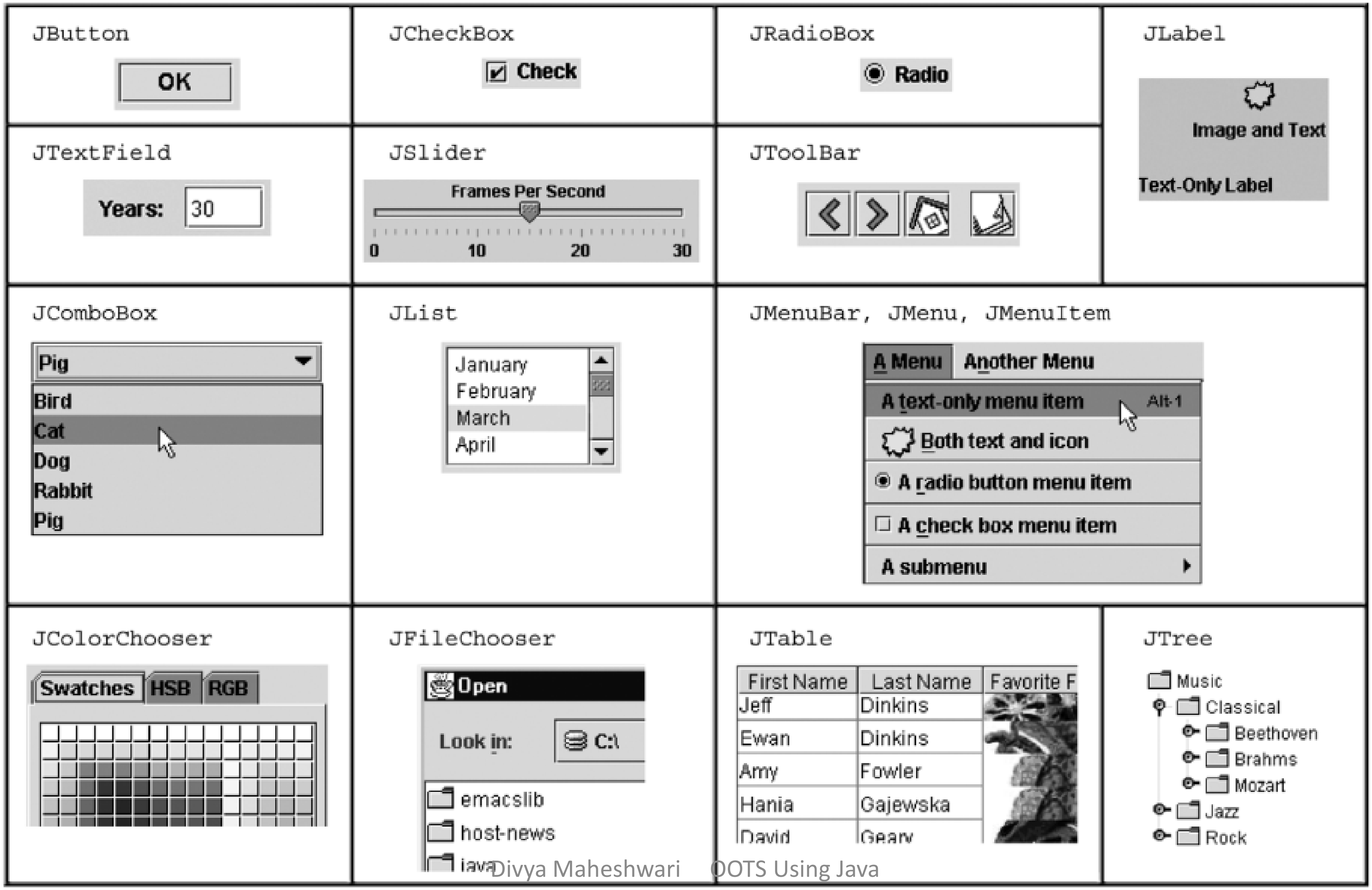


GUI terminology

- **window**: A first-class citizen of the graphical desktop.
  - Also called a *top-level container*.
  - examples: frame, dialog box, applet
- **component**: A GUI widget that resides in a window.
  - Also called *controls* in many other languages.
  - examples: button, text box, label
- **container**: A logical grouping for storing components.
  - examples: panel, box



Components


JButton
OK
JCheckBox
Check
JRadioBox
Radio
JLabel
Image and Text
Text-Only Label
JTextField
Years:
30
JSlider
Frames Per Second
0
10
20
30
JToolBar
JComboBox
Pig
Bird
Cat
Dog
Rabbit
Pig
JList
January
February
March
April
JMenuBar, JMenu, JMenuItem
A Menu
Another Menu
A text-only menu item
Alt-1
Both text and icon
A radio button menu item
A check box menu item
A submenu
JColorChooser
Swatches
HSB
RGB
JFileChooser
Open
Look in:
C:\
emacslib
host-news
java
JTable
First Name
Last Name
Favorite F
Jeff
Dinkins
Ewan
Dinkins
Amy
Fowler
Hania
Gajewska
David
Geary
JTree
Music
Classical
Beethoven
Brahms
Mozart
Jazz
Rock

Swing inheritance hierarchy

- Component (AWT)
  - Window
    - Frame
      - **JFrame** (Swing)
      - **JDialog**
  - Container
    - JComponent (Swing)

JButton	**JColorChooser**	**JFileChooser**
JComboBox	**JLabel**	**JList**
JMenuBar	**JOptionPane**	**JPanel**
JPopupMenu	**JProgressBar**	**JScrollbar**
JScrollPane	**JSlider**	**JSpinner**
JSplitPane	**JTabbedPane**	**JTable**
JToolbar	**JTree**	**JTextArea**
JTextField	**...**	

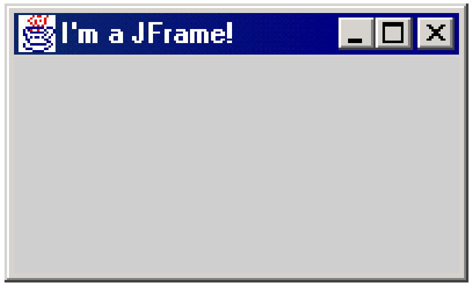
Component Properties

- Each has a `get` (or `is`) accessor and a `set` modifier method.
- examples: `getColor`, `setFont`, `setEnabled`, `isVisible`

name	type	description
background	`Color`	background color behind component
border	`Border`	border line around component
enabled	`boolean`	whether it can be interacted with
focusable	`boolean`	whether key text can be typed on it
font	`Font`	font used for text in component
foreground	`Color`	foreground color of component
height, width	`int`	component's current size in pixels
visible	`boolean`	whether component can be seen
tooltip text	`String`	text shown when hovering mouse
size, minimum / maximum / preferred size	`Dimension`	various sizes, size limits, or desired sizes that the component may take

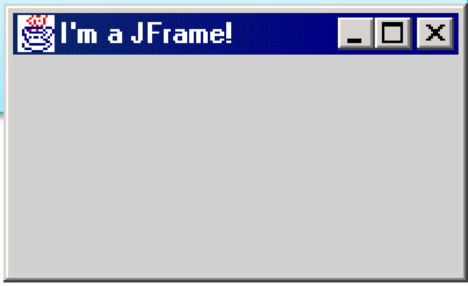
JFrame

a graphical window to hold other components



- `public JFrame()`
  `public JFrame(String title)`
  Creates a frame with an optional title.
  - Call `setVisible(true)` to make a frame appear on the screen after creating it.
- `public void add(Component comp)`
  Places the given component or container inside the frame.

More JFrame



- `public void setDefaultCloseOperation(int op)`
  Makes the frame perform the given action when it closes.
  - Common value passed: `JFrame.EXIT_ON_CLOSE`
  - If not set, the program will never exit even if the frame is closed.

- `public void setSize(int width, int height)`
  Gives the frame a fixed size in pixels.

- `public void pack()`
  Resizes the frame to fit the components inside it snugly.

JButton

a clickable region for causing actions to occur

- `public JButton(String text)`
  Creates a new button with the given string as its text.

- `public String getText()`
  Returns the text showing on the button.

- `public void setText(String text)`
  Sets button's text to be the given string.

GUI example

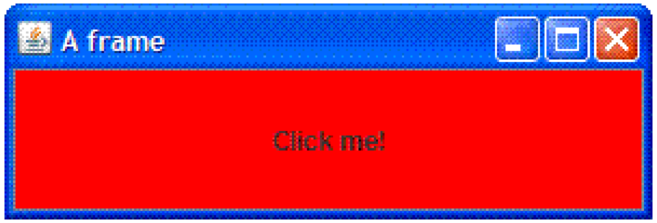
```
import java.awt.*;        // Where is the other button?
import javax.swing.*;

public class GuiExample1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(new Dimension(300, 100));
        frame.setTitle("A frame");

        JButton button1 = new JButton();
        button1.setText("I'm a button.");
        button1.setBackground(Color.BLUE);
        frame.add(button1);

        JButton button2 = new JButton();
        button2.setText("Click me!");
        button2.setBackground(Color.RED);
        frame.add(button2);

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



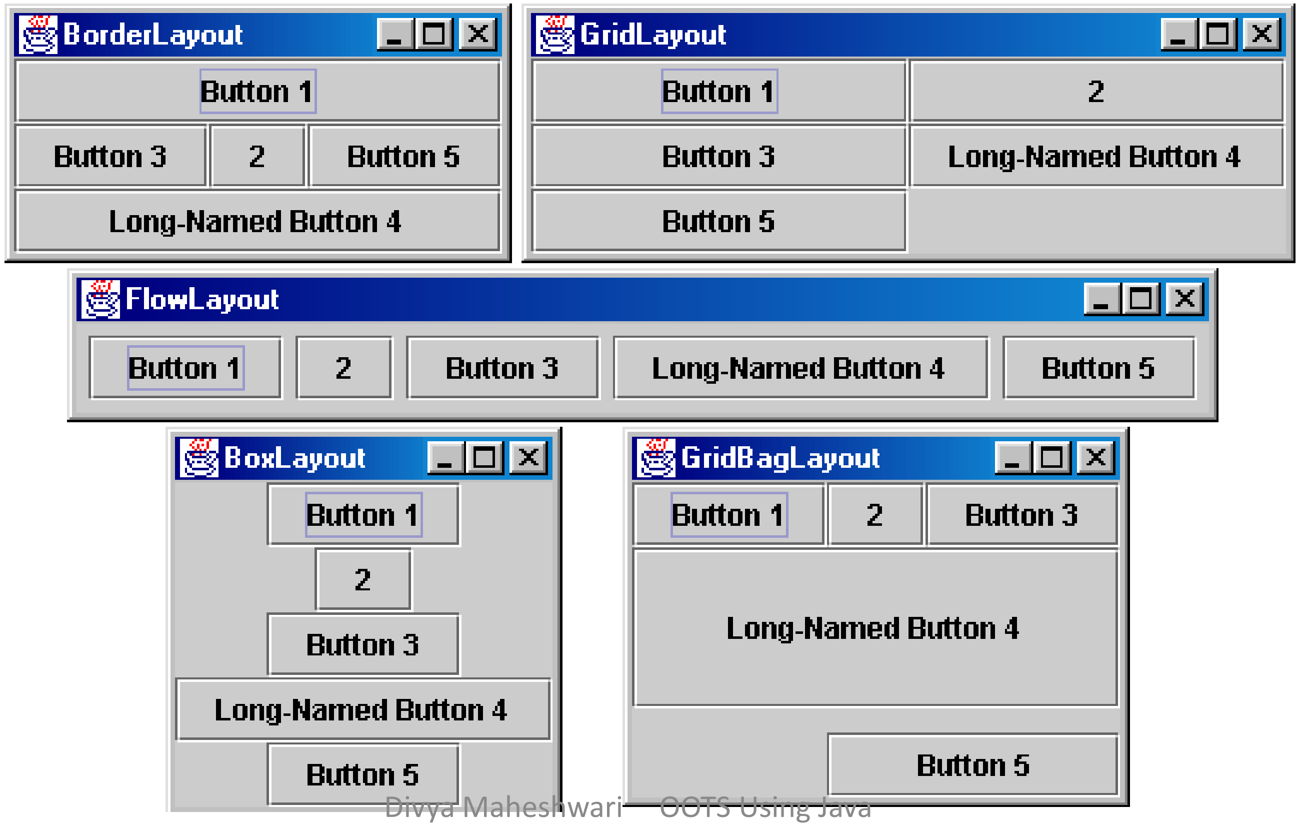
Sizing and positioning

How does the programmer specify where each component appears, how big each component should be, and what the component should do if the window is resized / moved / maximized / etc.?

- **Absolute positioning** (C++, C#, others):
  Programmer specifies exact pixel coordinates of every component.
  - "Put this button at (x=15, y=75) and make it 70x31 px in size."
- **Layout managers** (Java):
  Objects that decide where to position each component based on some general rules or criteria.
  - "Put these four buttons into a 2x2 grid and put these text boxes in a horizontal flow in the south part of the frame."

Containers and layout

- Place components in a *container*; add the container to a frame.
  - **container**: An object that stores components and governs their positions, sizes, and resizing behavior.



JFrame as container

A `JFrame` is a container. Containers have these methods:

- `public void` **`add`**`(Component comp)`
  `public void` **`add`**`(Component comp, Object info)`
  Adds a component to the container, possibly giving extra information about where to place it.

- `public void` **`remove`**`(Component comp)`

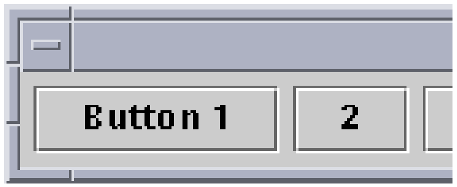
- `public void` **`setLayout`**`(LayoutManager mgr)`
  Uses the given layout manager to position components.

- `public void` **`validate`**`()`
  Refreshes the layout (if it changes after the container is onscreen).

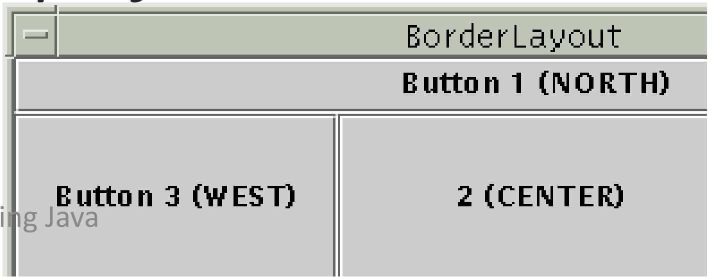
Preferred sizes

- Swing component objects each have a certain size they would "like" to be: Just large enough to fit their contents (text, icons, etc.).
  - This is called the *preferred size* of the component.
  - Some types of layout managers (e.g. `FlowLayout`) choose to size the components inside them to the preferred size.
  - Others (e.g. `BorderLayout`, `GridLayout`) disregard the preferred size and use some other scheme to size the components.

Buttons at preferred size:



Not preferred size:



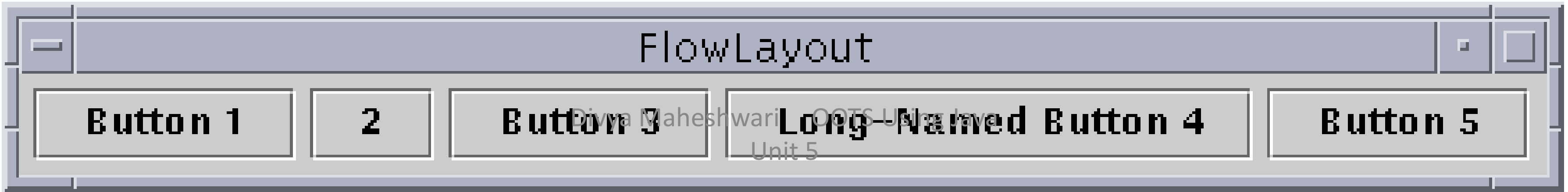
FlowLayout

`public FlowLayout()`

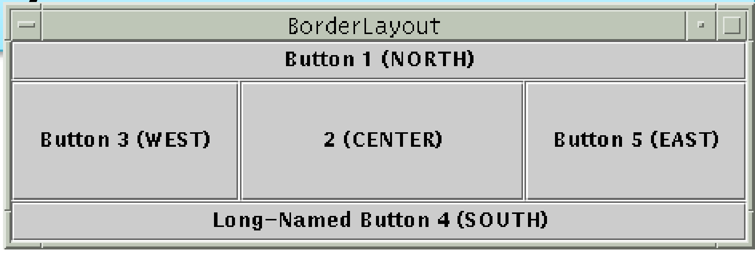
- treats container as a left-to-right, top-to-bottom "paragraph".
  - Components are given preferred size, horizontally and vertically.
  - Components are positioned in the order added.
  - If too long, components wrap around to the next line.

```
myFrame.setLayout(new FlowLayout());
myFrame.add(new JButton("Button 1"));
```

  - The default layout for containers other than `JFrame` (seen later).



BorderLayout



`public BorderLayout()`

- Divides container into five regions:
  - `NORTH` and `SOUTH` regions expand to fill region horizontally, and use the component's preferred size vertically.
  - `WEST` and `EAST` regions expand to fill region vertically, and use the component's preferred size horizontally.
  - `CENTER` uses all space not occupied by others.

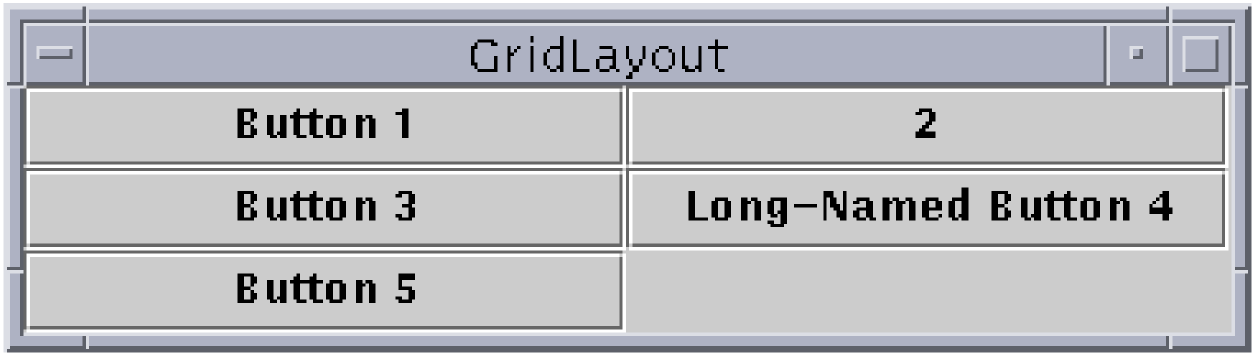
```
myFrame.setLayout(new BorderLayout());
myFrame.add(new JButton("Button 1"), BorderLayout.NORTH);
```

  - This is the default layout for a `JFrame`.

GridLayout

`public GridLayout(int rows, int columns)`

- Treats container as a grid of equally-sized rows and columns.
- Components are given equal horizontal / vertical size, disregarding preferred size.
- Can specify 0 rows or columns to indicate expansion in that direction as needed.

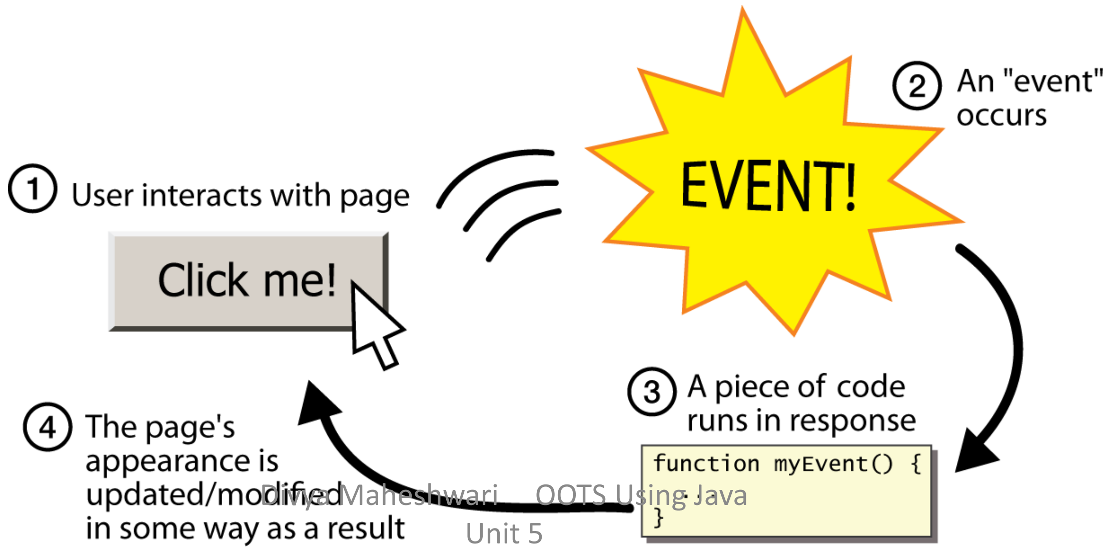


Event Listeners

Graphical events

- **event**: An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components.

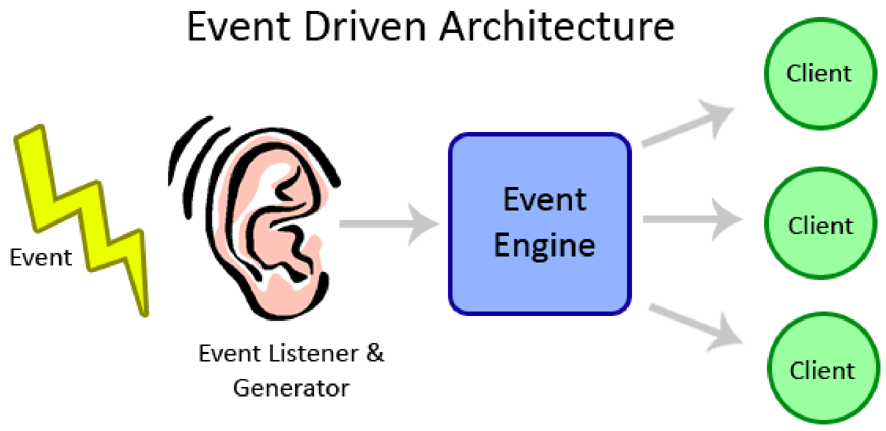
- **listener**: An object that waits for events and responds to them.
  - To handle an event, attach a *listener* to a component.
  - The listener will be notified when the event occurs (e.g. button click).



① User interacts with page
Click me!
EVENT!
② An "event" occurs
③ A piece of code runs in response
function myEvent() {
}
④ The page's appearance is updated/modified in some way as a result

Event-driven programming

- **event-driven programming**: A style of coding where a program's overall flow of execution is dictated by events.
  - Rather than a central "main" method that drives execution, the program loads and waits for user input events.
  - As each event occurs, the program runs particular code to respond.
  - The overall flow of what code is executed is determined by the series of events that occur, not a pre-determined order.



Event hierarchy

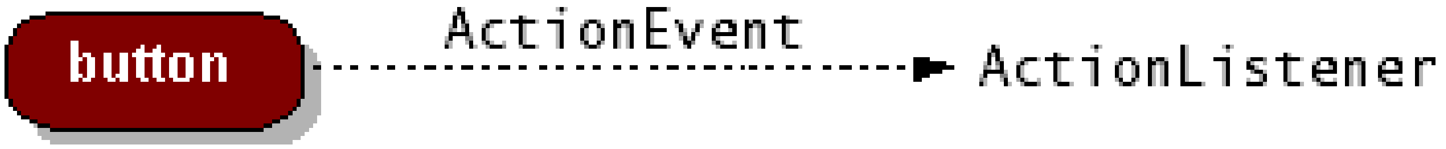
- EventObject
  - AWTEvent (AWT)
    - **ActionEvent**
    - TextEvent
    - ComponentEvent
      - FocusEvent
      - WindowEvent
      - InputEvent
        - KeyEvent
        - MouseEvent

- EventListener
  - AWTEventListener
  - **ActionListener**
  - TextListener
  - ComponentListener
  - FocusListener
  - WindowListener

  - KeyListener
  - MouseListener

Action events

- **action event**: An action that has occurred on a GUI component.
  - The most common, general event type in Swing. Caused by:
    - button or menu clicks,
    - check box checking / unchecking,
    - pressing Enter in a text field, ...
  - Represented by a class named `ActionEvent`
  - Handled by objects that implement interface `ActionListener`



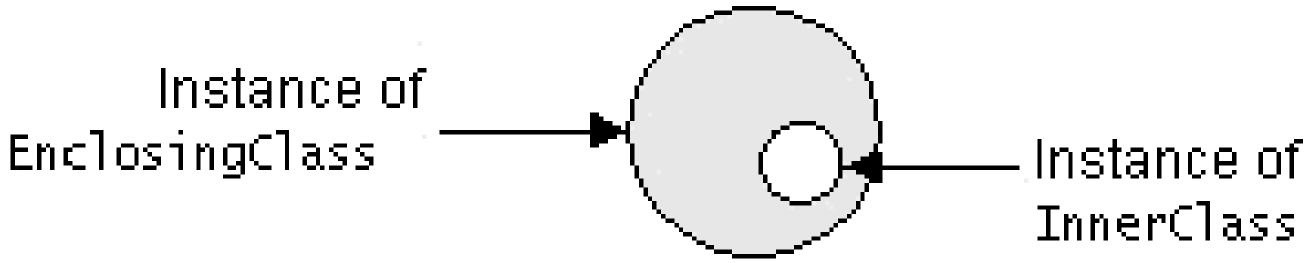
Implementing a listener

```
public class name implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        code to handle the event;
    }
}
```

- `JButton` and other graphical components have this method:
  - `public void addActionListener(ActionListener al)` Attaches the given listener to be notified of clicks and events that occur on this component.

Nested classes

- **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);
```

AWT

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

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea RadioButton, CheckBox, Choice, List etc.

AWT

Why AWT is platform independent?

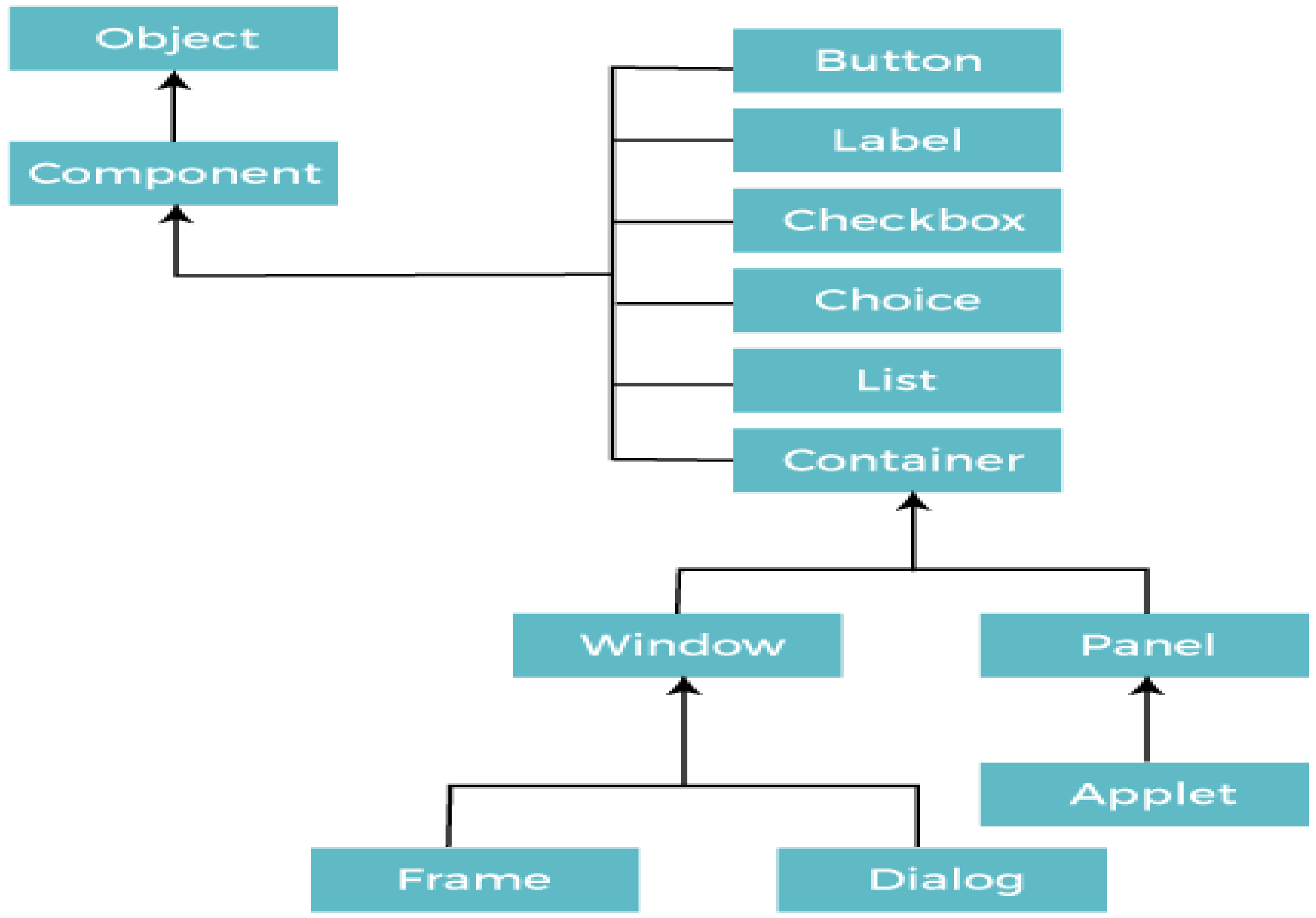
An AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix.

The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

AWT Hierarchy

- The hierarchy of Java AWT classes are given below.



AWT Hierarchy

- Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container. (Read the components from class notes)

- Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

AWT Hierarchy

Types of containers:

There are four types of containers in Java AWT:

Window

Panel

Frame

Dialog

1.Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

2.Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

3.Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

AWT Hierarchy

4. Dialog- Th Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the window class. Unlike the frame it does not have maximize and minimize buttons.

Applet- An applet is a java program that runs in a web browser and embedded into a web page. They perform small task like like sound play, animation. It is used to make a web page more dynamic. If we want to perform graphic display, animations, sound play on web page then for that we will write java programs known as applet that will run in web browser.

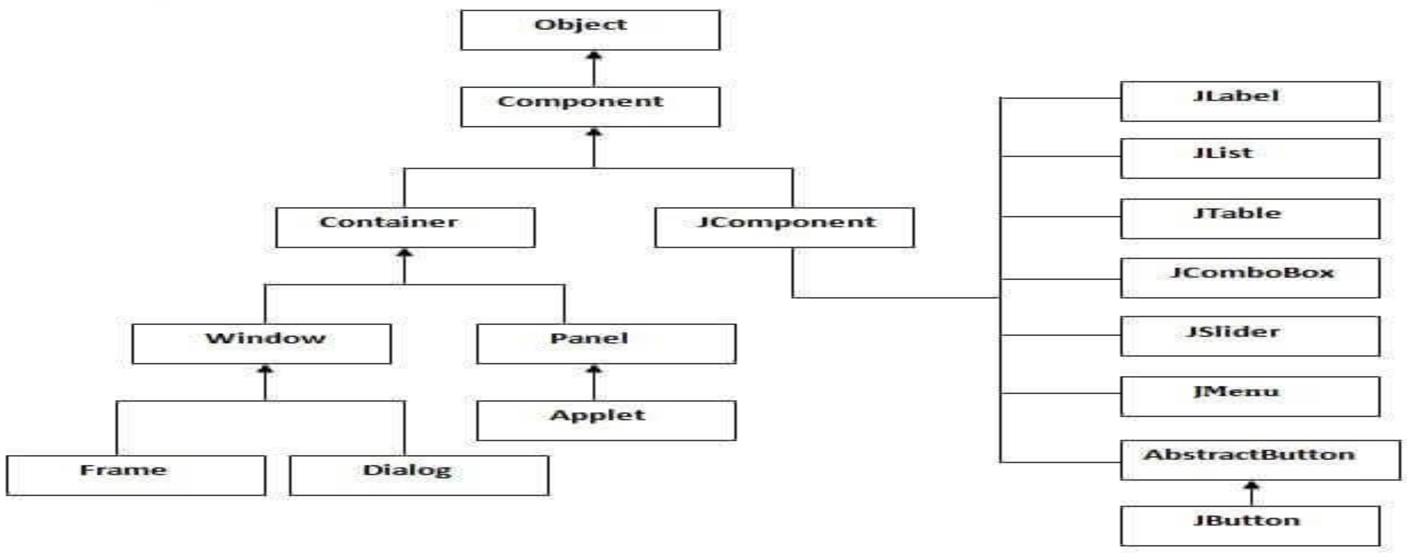
Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Hierarchy of Java Swing Classes



Swing

The methods of Component class are widely used in java swing that are given below.

public void add(Component c)-add a component on another component.

public void setSize(int width,int height)- sets size of the component.

public void setLayout(LayoutManager m)-sets the layout manager for the component.

public void setVisible(boolean b)- sets the visibility of the component. It is by default false.

What is JFC?

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications. Standard Edition (J2SE) to support building graphics user interface (GUI) and graphics functionality for client applications that will run on popular platforms such as Microsoft Windows, Linux, and Mac OSX.

Difference between AWT and Swing

AWT	Swing
AWT components are heavyweight components	Swing components are lightweight components
AWT doesn't support pluggable look and feel	Swing supports pluggable look and feel
AWT programs are not portable	Swing programs are portable
AWT is old framework for creating GUIs	Swing is new framework for creating GUIs
AWT components require java.awt package	Swing components require javax.swing package
AWT supports limited number of GUI controls	Swing provides advanced GUI controls like Jtable, JTabbedPane etc
More code is needed to implement AWT controls functionality	Less code is needed to implement swing controls functionality
AWT doesn't follow MVC	Swing follows MVC

QUESTIONS

Q. Why Swing components are called lightweight components?

Q What are differences between Swing and AWT?

Q What do heavy weight components mean?

Q. What is Event-Dispatcher-Thread (EDT) in Swing?

Q. Does Swing is thread safe?

Q. What do heavy weight components mean?

Q. How can a GUI component handle its own events?

Q. What do you understand by Components and Containers.

Q. What interface is extended by AWT event listener?

Q.What is the difference between a Window and a Frame?

QUESTIONS

Q. **What is the difference between the paint() and repaint() methods?**

Ans- The paint() method supports painting via a Graphics object.

The repaint() method is used to cause paint() to be invoked by the AWT painting thread.

Q. **What Advantage Do Java's Layout Managers Provide Over Traditional Windowing Systems?**

Ans -Java uses layout managers to lay out components in a consistent manner across all windowing platforms. Since Java's layout managers aren't tied to absolute sizing and positioning, they are able to accommodate platform specific differences among windowing systems.

Q. **Why Collection doesn't extend the Cloneable and Serializable interfaces?**

Ans- The Collection interface in Java specifies a group of objects called elements. The maintainability and ordering of elements is completely dependent on the concrete implementations provided by each of the Collection. Thus, there is no use of extending the Cloneable and Serializable interfaces.

QUESTIONS

Q. Can you add a null element into a TreeSet or HashSet?

In HashSet, only one null element can be added but in TreeSet it can’t be added as it makes use of NavigableMap for storing the elements. This is because the NavigableMap is a subtype of SortedMap that doesn’t allow null keys. So, in case you try to add null elements to a TreeSet, it will throw a NullPointerException.

Q. Can You Store A Duplicate Key In Hashmap?

Ans- No, you cannot insert duplicate keys in HashMap, it doesn't allow duplicate keys. If you try to insert an existing key with new or same value then it will override the old value but size of HashMap will not change i.e. it will remain same. This is one of the reason when you get all keys from the HashMap by calling keySet() it returns a Set, not a Collection because Set doesn't allow duplicates.

QUESTIONS

Q. What Is Type Erasure?

Ans- Generics are used for tighter type checks at compile time and to provide a generic programming. To implement generic behaviour, java compiler apply type erasure. Type erasure is a process in which compiler replaces a generic parameter with actual class or bridge method. In type erasure, compiler ensures that no extra classes are created and there is no runtime overhead.

Type Erasure rules

- Replace type parameters in generic type with their bound if bounded type parameters are used.
- Replace type parameters in generic type with Object if unbounded type parameters are used.
- Insert type casts to preserve type safety.
- Generate bridge methods to keep polymorphism in extended generic types.

QUESTIONS

It’s important to understand type erasure. Otherwise, a developer might get confused and think they’d be able to get the type at runtime :

```
public foo(Consumer<T> consumer) {
   Type type = consumer.getGenericTypeParameter()
}
```

The above example is a pseudo code equivalent of what things might look like without type erasure, but unfortunately, it is impossible. Once again, the generic type information is not available at runtime.

Java 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

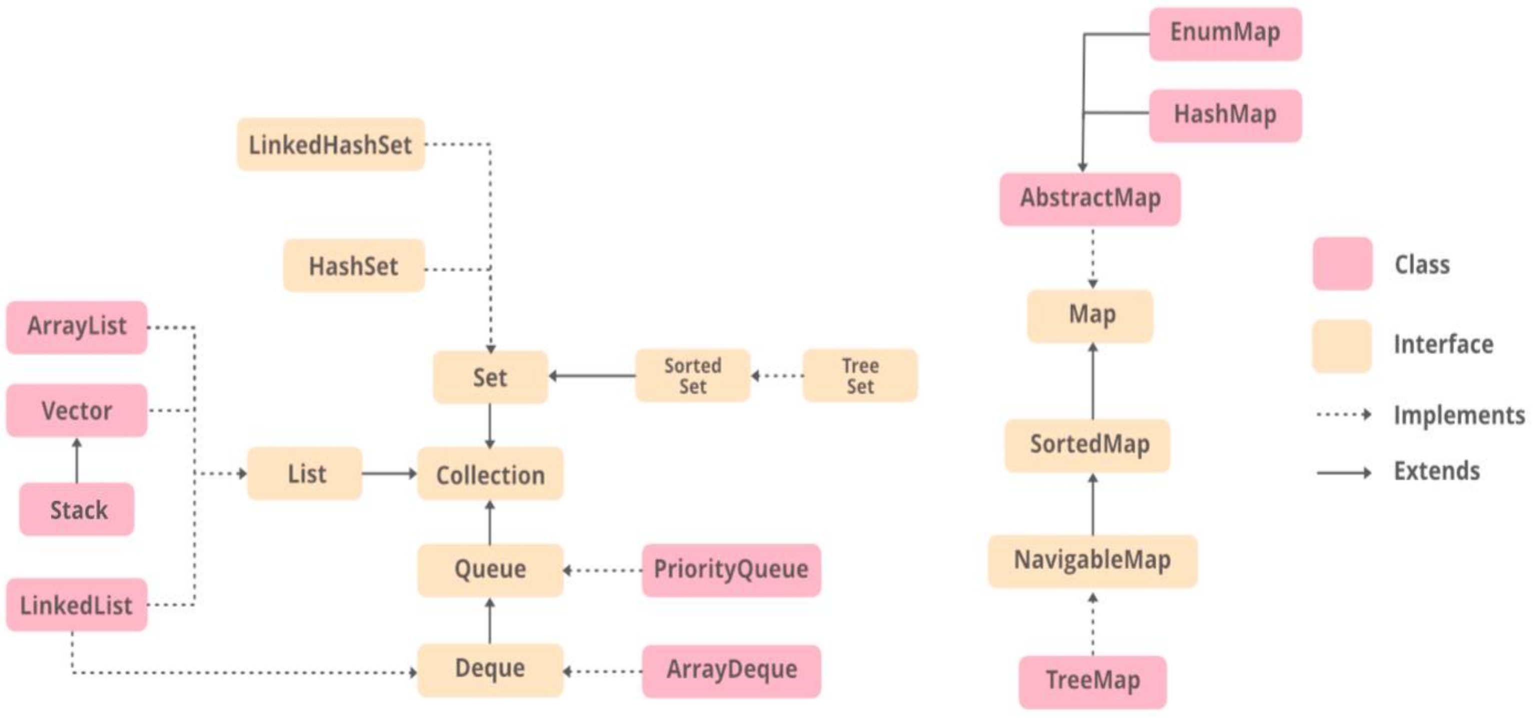
- Unified architecture for representing and manipulating collections.
- A collections framework contains three things
  - » Interfaces
  - » Implementations
  - » Algorithms

Collections Framework

Need for a Separate Collection Framework

Before the Collection Framework(or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were **Arrays** or **Vectors**, or **Hashtables**. All of these collections had no common interface. Therefore, though the main aim of all the collections is the same, the implementation of all these collections was defined independently and had no correlation among them. And also, it is very difficult for the users to remember all the different **methods**, syntax, and **constructors** present in every collection class.

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

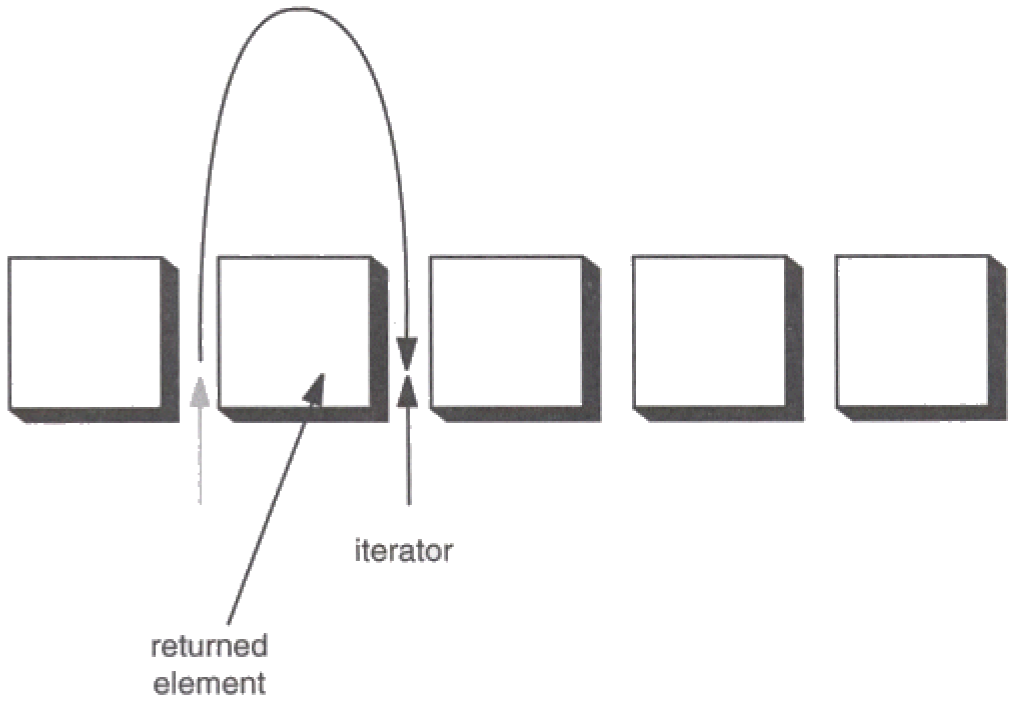


Figure 2–3: Advancing an iterator

List Interface

This is a child interface of the collection interface. This interface is dedicated to the data of the list type in which we can store all the ordered collection of the objects.

This also allows duplicate data to be present in it. This list interface is implemented by various classes like ArrayList, Vector, Stack, etc. Since all the subclasses implement the list, we can instantiate a list object with any of these classes. For example,

```
List <T> al = new ArrayList<> ();
List <T> ll = new LinkedList<> ();
List <T> v = new Vector<> ();
```

ArrayList

ArrayList provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

The size of an ArrayList is increased automatically if the collection grows or shrinks if the objects are removed from the collection. Java ArrayList allows us to randomly access the list.

ArrayList can not be used for primitive types, like int, char, etc. We will need a wrapper class for such cases.

LinkedList

LinkedList class is an implementation of the LinkedList data structure which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part.

The elements are linked using pointers and addresses. Each element is known as a node.

vector

A vector provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

This is identical to ArrayList in terms of implementation. However, the primary difference between a vector and an ArrayList is that a Vector is synchronized and an ArrayList is non-synchronized.

stack

Stack class models and implements the Stack data structure.

The class is based on the basic principle of *last-in-first-out*. In addition to the basic push and pop operations, the class provides three more functions of **empty, search and peek.**

The class can also be referred to as the subclass of Vector.

_________ is the given classes present in System.Collection.Generic.namespace.

Ans- Stack

Queue Interface

As the name suggests, a queue interface maintains the FIFO(First In First Out) order similar to a real-world queue line. This interface is dedicated to storing all the elements where the order of the elements matter.

For example, whenever we try to book a ticket, the tickets are sold on a first come first serve basis. Therefore, the person whose request arrives first into the queue gets the ticket.

There are various classes like_PriorityQueue, ArrayDeque, etc. Since all these subclasses implement the queue, we can instantiate a queue object with any of these classes.

```
Queue <T> pq = new PriorityQueue<> ();
Queue <T> ad = new ArrayDeque<> ();
```

PriorityQueue

A PriorityQueue is used when the objects are supposed to be processed based on the priority.

It is known that a queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority and this class is used in these cases. The PriorityQueue is based on the priority heap.

The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

Deque Interface

This is a very slight variation of the **queue data structure**. **Deque**, also known as a double-ended queue, is a data structure where we can add and remove the elements from both ends of the queue. This interface extends the queue interface.

The class which implements this interface is ArrayDeque.

Since ArrayDeque class implements the Deque interface, we can instantiate a deque object with this class.

Deque<T> ad = new ArrayDeque<> ();

ArrayDeque class

ArrayDeque class which is implemented in the collection framework provides us with a way to apply resizable-array.

This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue.

Array deques have no capacity restrictions and they grow as necessary to support usage.

Set Interface

A set is an unordered collection of objects in which duplicate values cannot be stored.

This collection is used when we wish to avoid the duplication of the objects and wish to store only the unique objects.

This set interface is implemented by various classes like HashSet, TreeSet, LinkedHashSet, etc. Since all the subclasses implement the set, we can instantiate a set object with any of these classes.

```
Set<T> hs = new HashSet<> ();
Set<T> lhs = new LinkedHashSet<> ();
Set<T> ts = new TreeSet<> ();
```

Q __________ is the class that implements Set interface.

Ans- HashSet

HashSet

The HashSet class is an inherent implementation of the hash table data structure.

The objects that we insert into the HashSet do not guarantee to be inserted in the same order. The objects are inserted based on their hashcode.

This class also allows the insertion of NULL elements.

LinkedHashSet

A LinkedHashSet is very similar to a HashSet.

The difference is that this uses a doubly linked list to store the data and retains the ordering of the elements.

Q The unique feature of LinkedHashSet is __________ .

Ans- It maintains the insertion order and guarantees uniqueness

SortedSet Interface

This interface is very similar to the set interface. The only difference is that this interface has extra methods that maintain the ordering of the elements.

The sorted set interface extends the set interface and is used to handle the data which needs to be sorted.

The class which implements this interface is TreeSet. Since this class implements the SortedSet, we can instantiate a SortedSet object with this class.

SortedSet<T> ts = new TreeSet<> ();

TreeSet

The TreeSet class uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided.

This must be consistent with equals if it is to correctly implement the Set interface.

It can also be ordered by a Comparator provided at set creation time, depending on which constructor is used.

Q The difference between TreeSet and SortedSet _________ .

Ans- SortedSet is an interface; TreeSet is a concrete class

List and set

List	Set
1. The List is an ordered sequence.	1. The Set is an unordered sequence.
2. List allows duplicate elements	2. Set doesn’t allow duplicate elements.
3. Elements by their position can be accessed.	3. Position access to elements is not allowed.
4. Multiple null elements can be stored.	4. Null element can store only once.
5. List implementations are ArrayList, LinkedList, Vector, Stack	5. Set implementations are HashSet, LinkedHashSet.

Map Interface

A map is a data structure that supports the key-value pair mapping for the data. This interface doesn't support duplicate keys because the same key cannot have multiple mappings.

A map is useful if there is data and we wish to perform operations on the basis of the key. This map interface is implemented by various classes like HashMap, TreeMap, etc.

Since all the subclasses implement the map, we can instantiate a map object with any of these classes.

```
Map<T> hm = new HashMap<> ();
Map<T> tm = new TreeMap<> ();
```

Map Interface

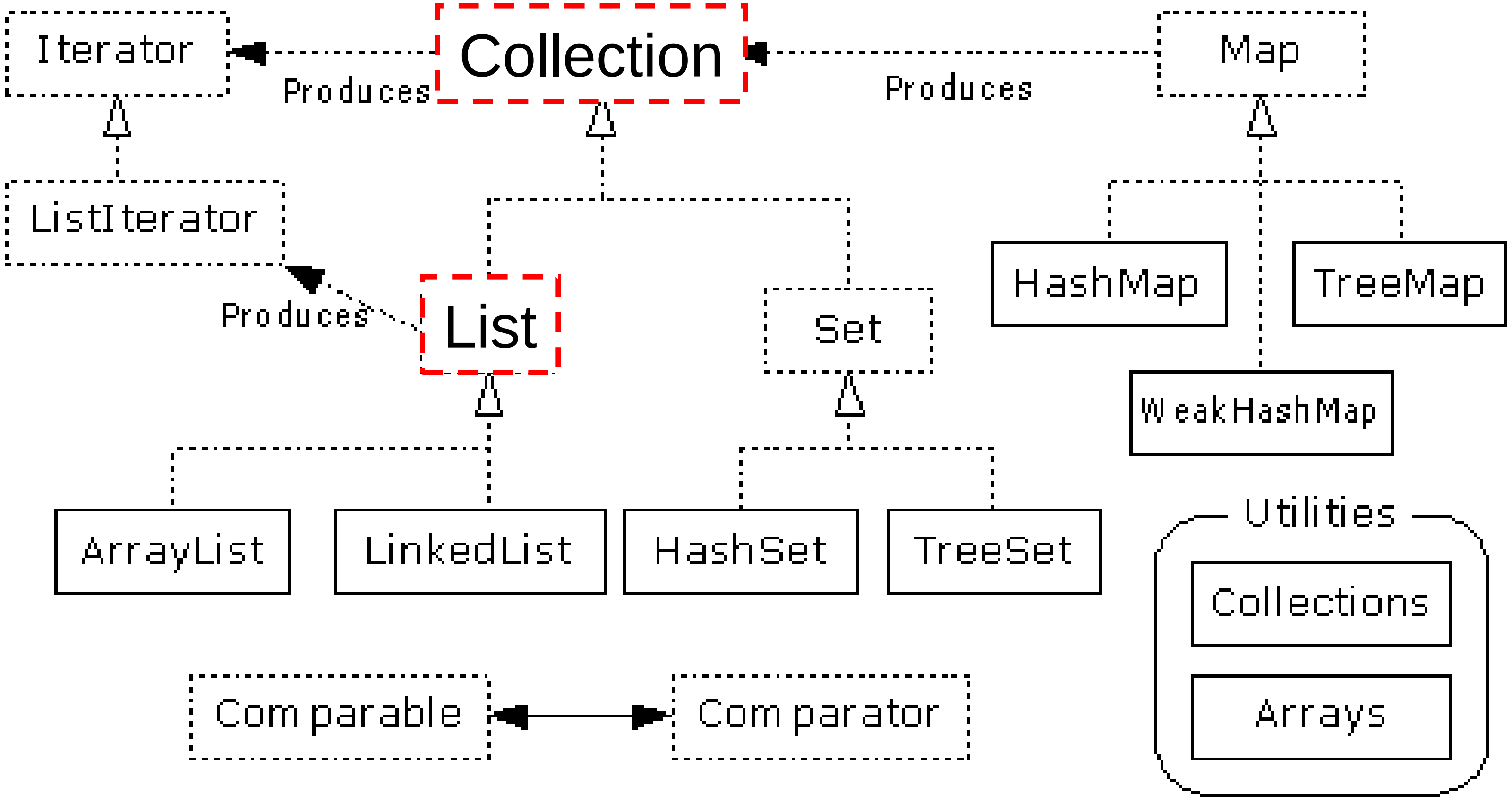
HashMap provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value in a HashMap, we must know its key.

HashMap uses a technique called Hashing. Hashing is a technique of converting a large String to a small String that represents the same String so that the indexing and search operations are faster.

HashSet also uses HashMap internally.

Java.util.Map interface provides the capability to store objects using a key-value pair.

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.

NIET Greater Noida
GET FUTURE READY

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()

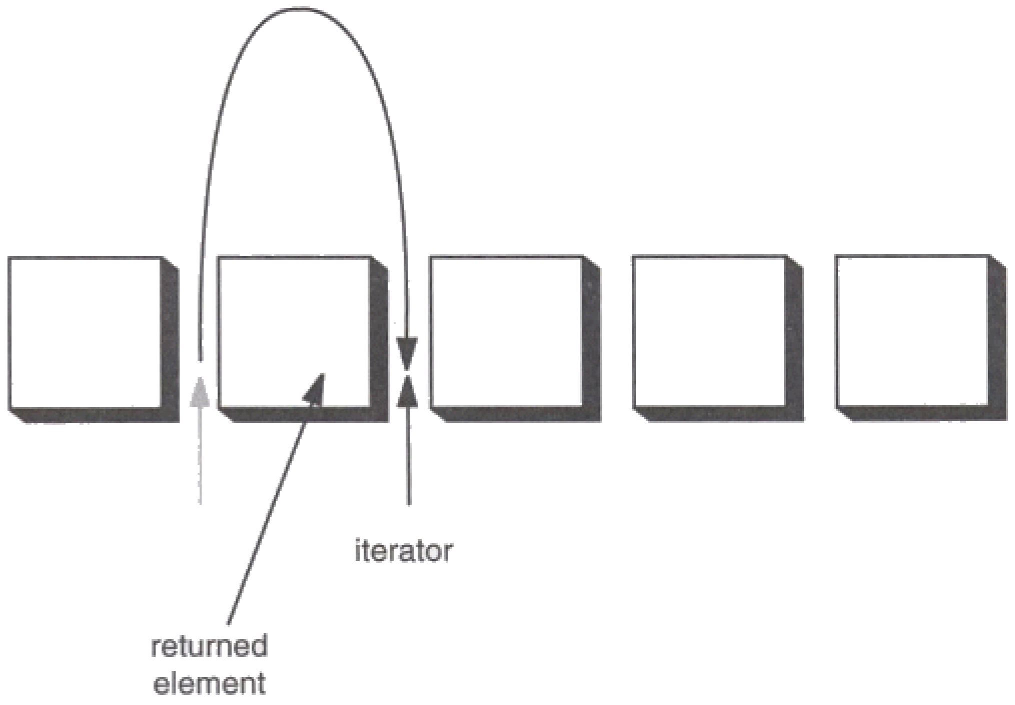
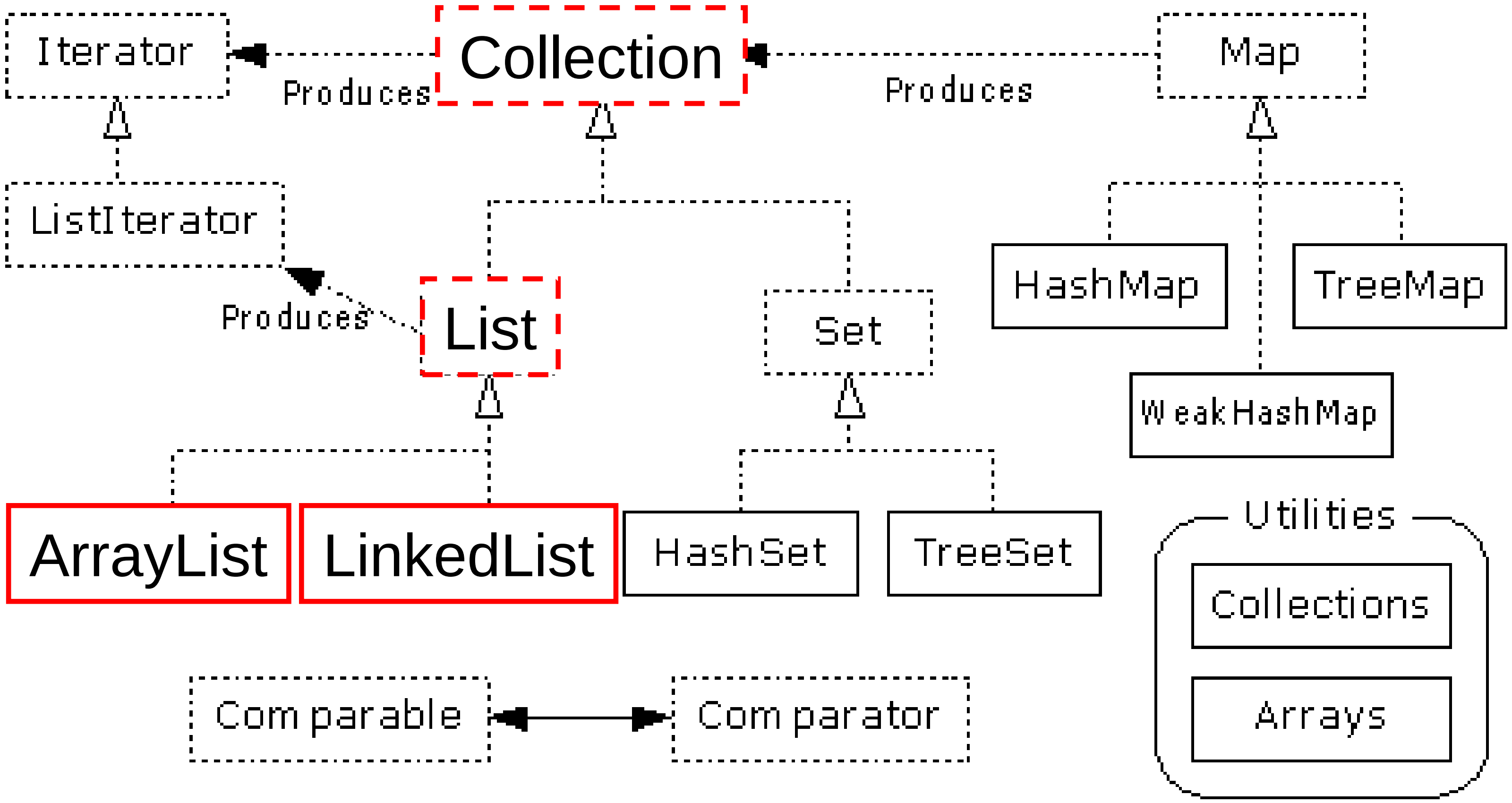


Figure 2–3: Advancing an iterator

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];
}
```

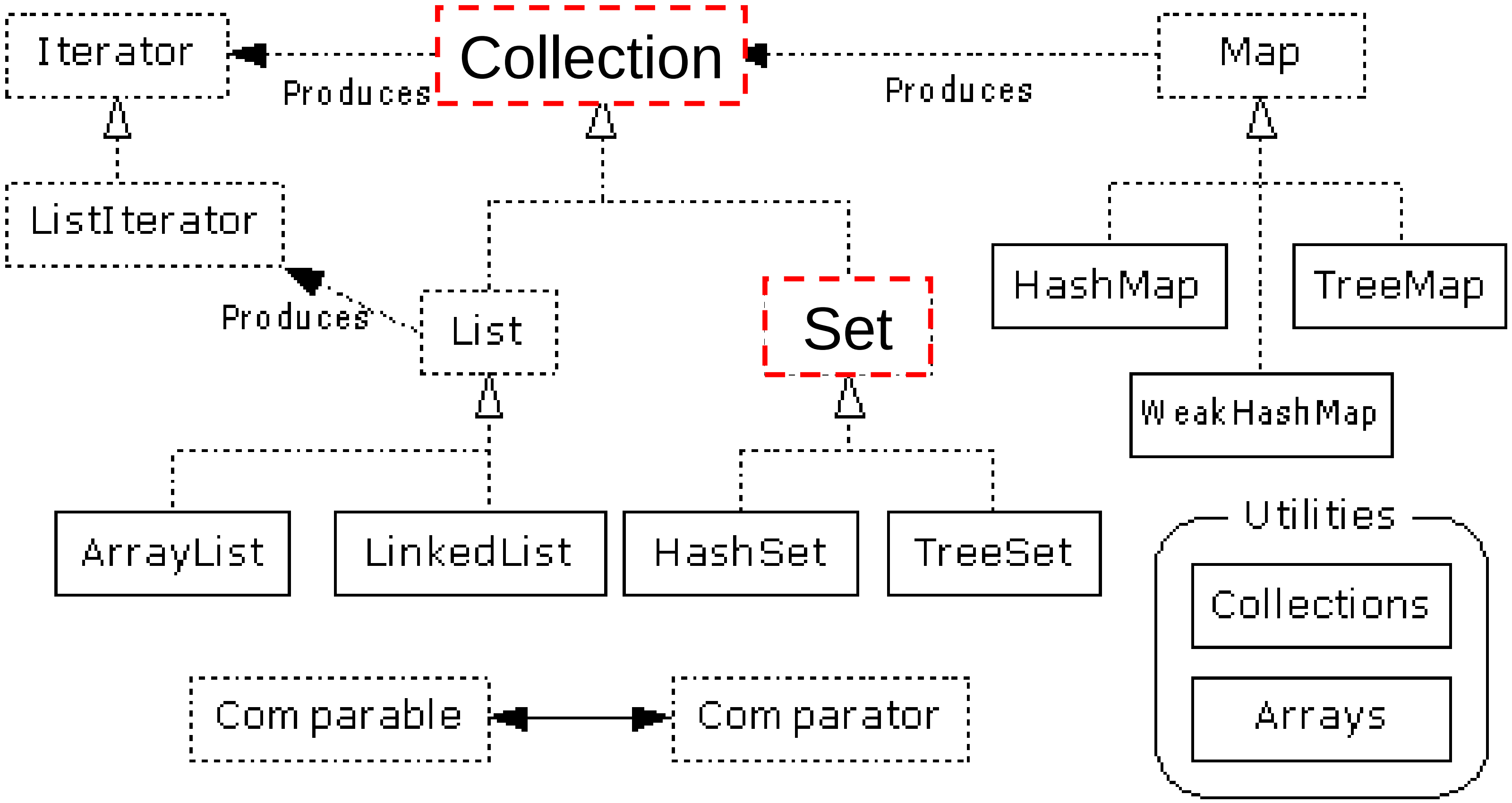
ArrayList methods

- 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

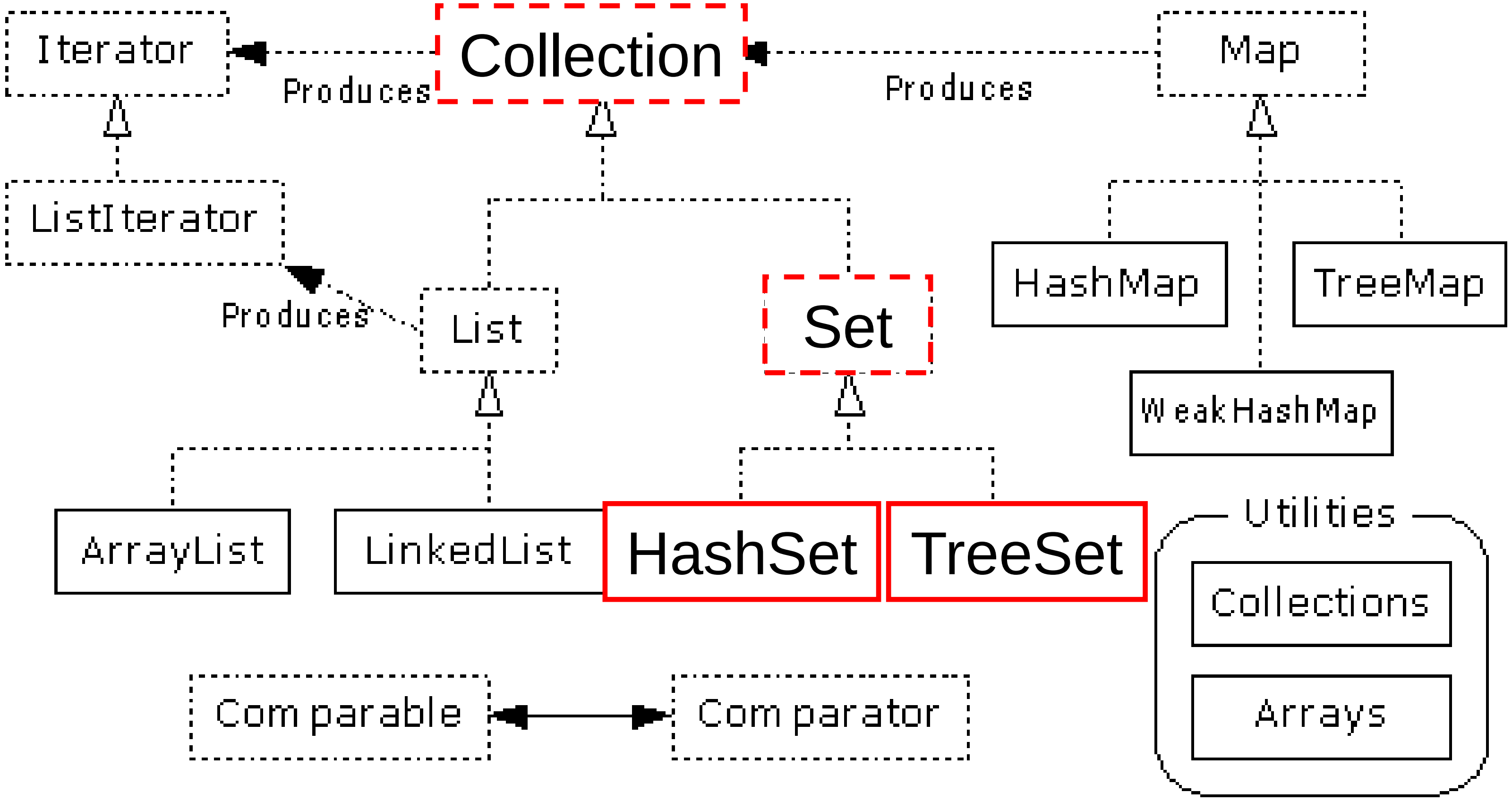
Set Interface Context



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



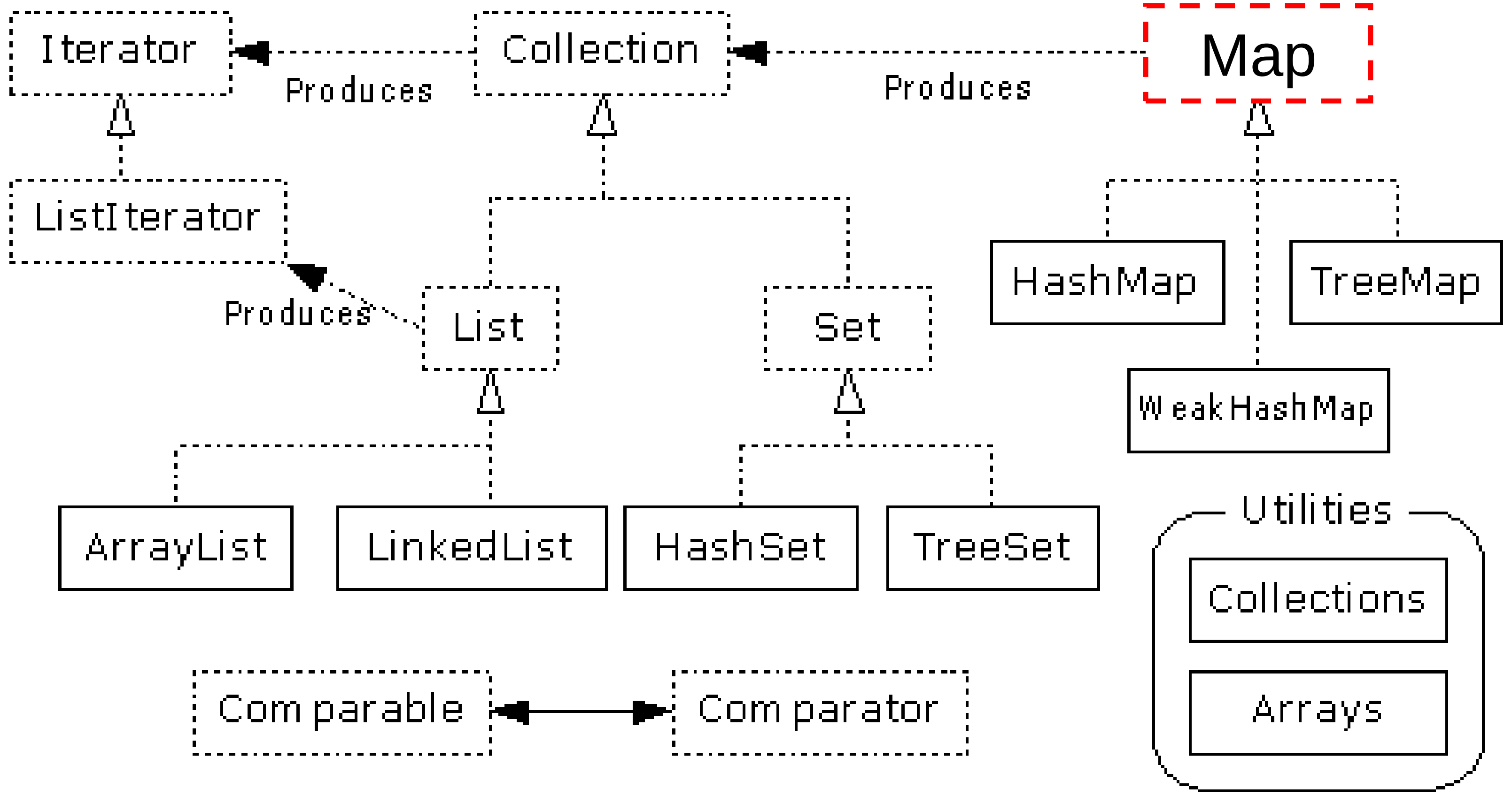
HashSet

- 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

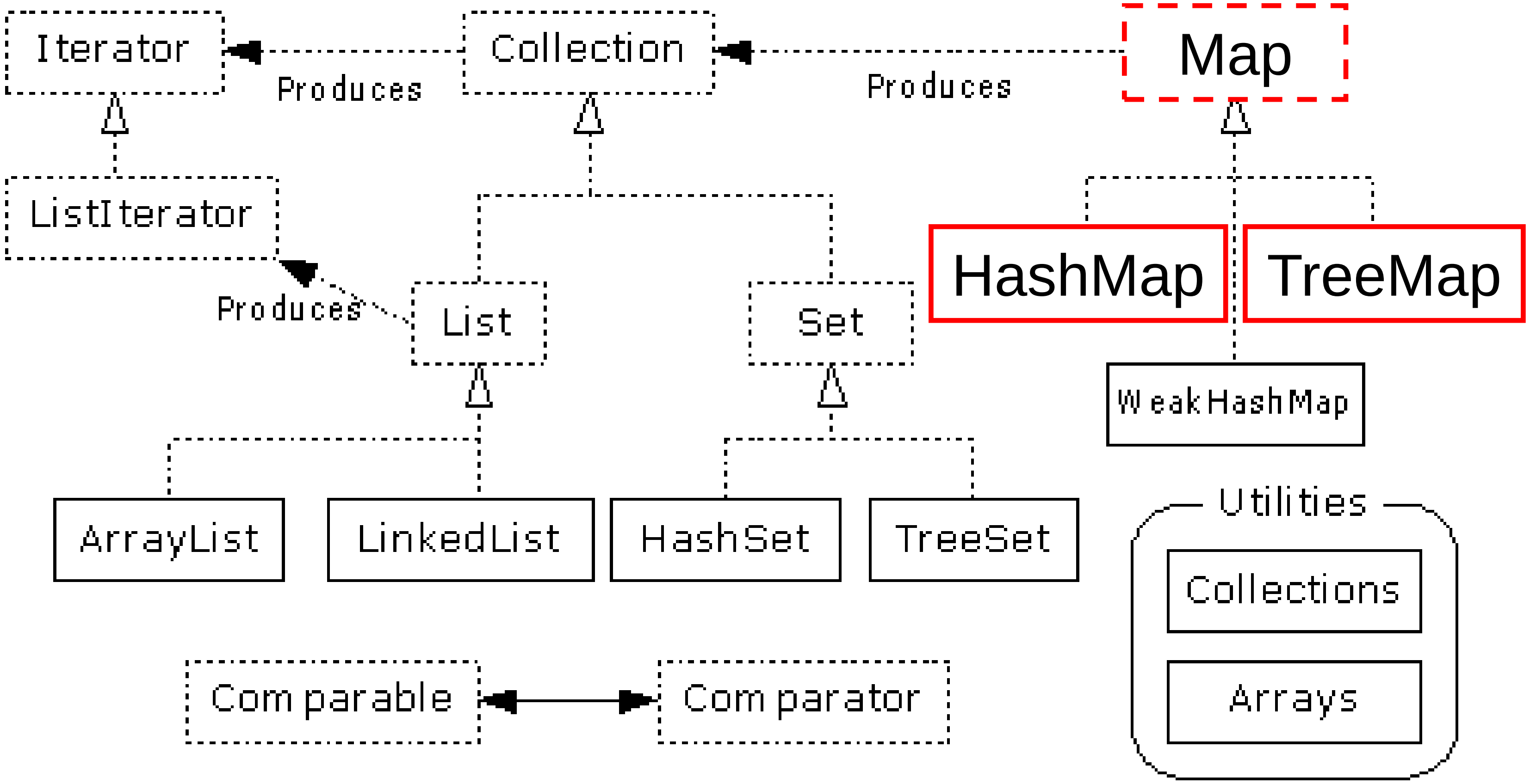
TreeSet

- 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



HashMap and TreeMap Context

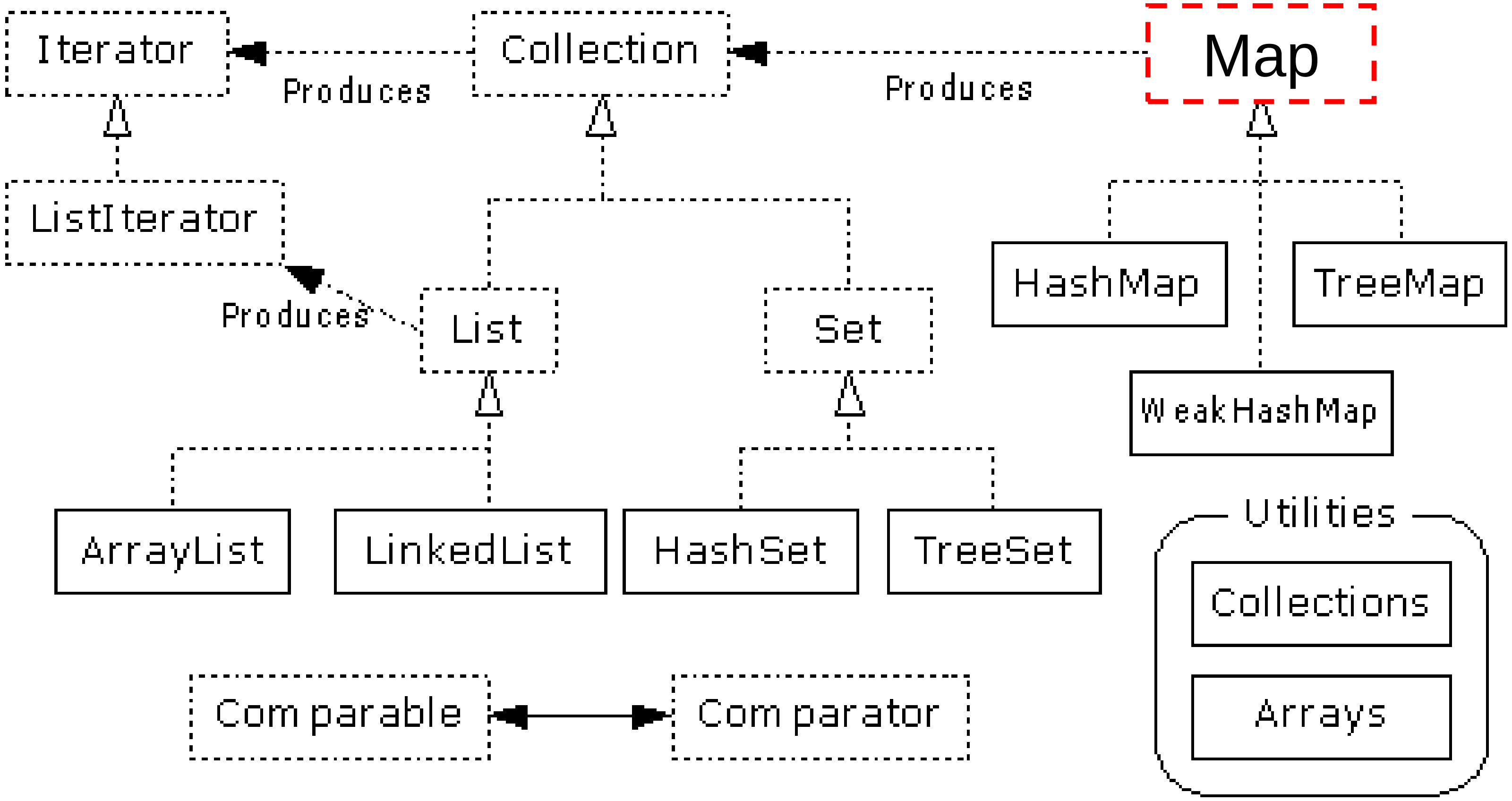


HashMap and TreeMap

- 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))*

Map Interface Context



Map Interface

- 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

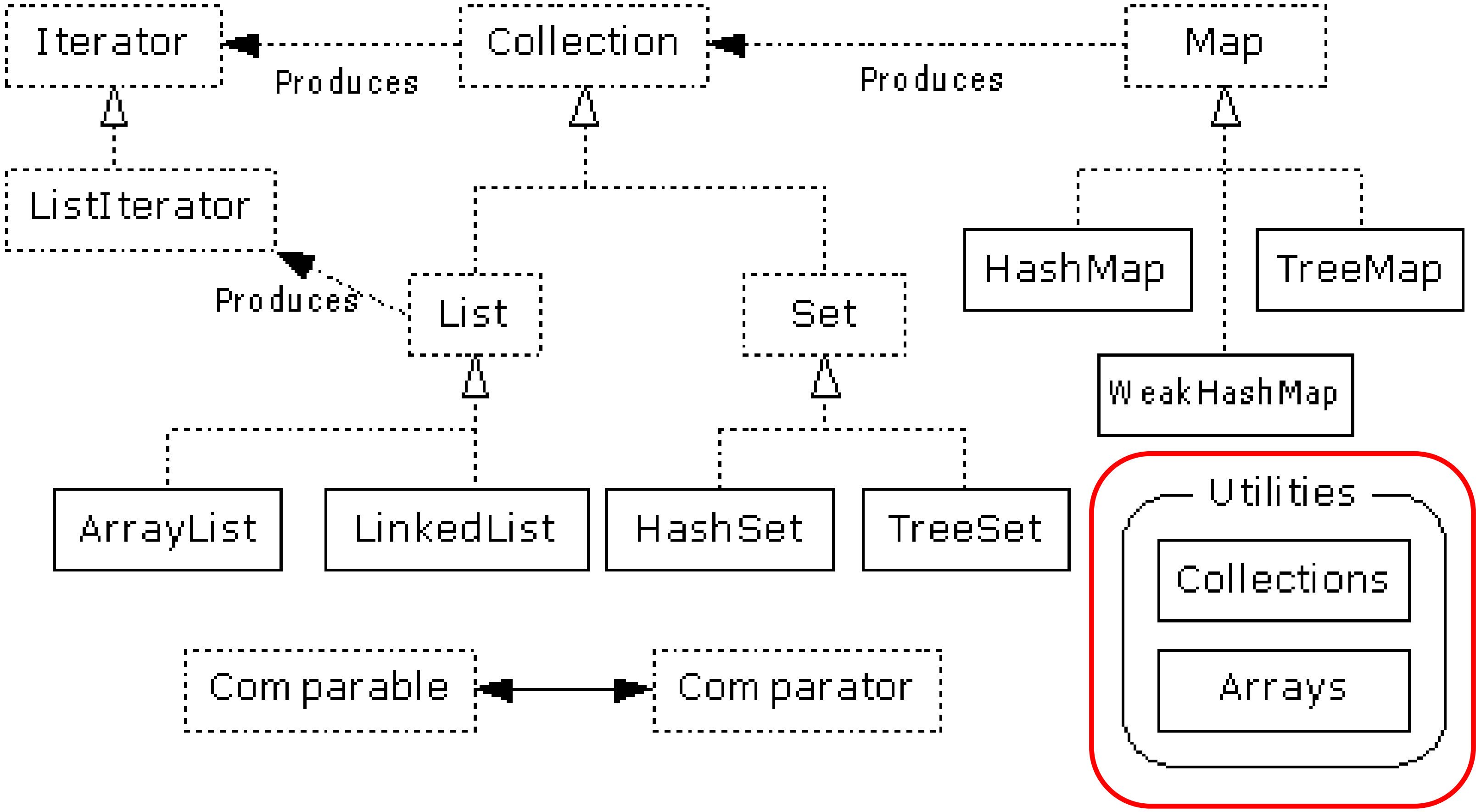
- A means of iterating over the keys and values in a Map
- **`Set keySet()`**
  - » returns the Set of keys contained in the Map
- **`Collection values()`**
  - » returns the Collection of values contained in the Map. This Collection is not a Set, as multiple keys can map to the same value.
- **`Set 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.

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

- 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

System properties

- Java VM maintains set of properties that define system environment
  - » Set when VM is initialized
  - » Includes information about current user, VM version, Java environment, and OS configuration

```
Properties prop = System.getProperties();
Enumeration e = prop.propertyNames();
while (e.hasMoreElements()) {
   String key = (String) e.nextElement();
   System.out.println(key + " value is " +
        prop.getProperty(key));
}
```

Method References

Java provides a new feature called method reference in Java 8. Method reference is used to refer method of functional interface. It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference.

Types of Method References

There are following types of method references in java:

1)Reference to a static method.

2)Reference to an instance method.

3)Reference to a constructor.

1) Reference to a Static Method

You can refer to static method defined in the class. Following is the syntax which describe the process of referring static method in Java.

Syntax

ContainingClass::staticMethodName

Method References

2) Reference to an Instance Method

like static methods, you can refer instance methods also. In the following example, we are describing the process of referring the instance method.

Syntax

containingObject::instanceMethodName

3) Reference to a Constructor

You can refer a constructor by using the new keyword. Here, we are referring constructor with the help of functional interface.

Syntax

ClassName::**new**

Example-

```
@FunctionalInterface
interface MyInterface
{
Hello display(String say);
}
class Hello{
 public Hello(String say)
{
System.out.print(say);
```

Method References

```
  }
 }
public class Example
 {
public static void main(String[] args) {
 //Method reference to a constructor
MyInterface ref = Hello::new;
ref.display("Hello World!");
 }
 }
```

Method References

```
  }
 }
public class Example
 {
public static void main(String[] args) {
 //Method reference to a constructor
MyInterface ref = Hello::new;
ref.display("Hello World!");
 }
 }
```

Method References

2) Reference to an Instance Method

like static methods, you can refer instance methods also.

Syntax

containingObject::instanceMethodName

3) Reference to a Constructor

You can refer a constructor by using the new keyword. Here, we are referring constructor with the help of functional interface.

Syntax

ClassName::**new**

_________ are the type of method references.

A. Reference to a static method.

B. Reference to an instance method.

C. Reference to a constructor.

D. All of the mentioned

Wrapper Class

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
- Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing**.**

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

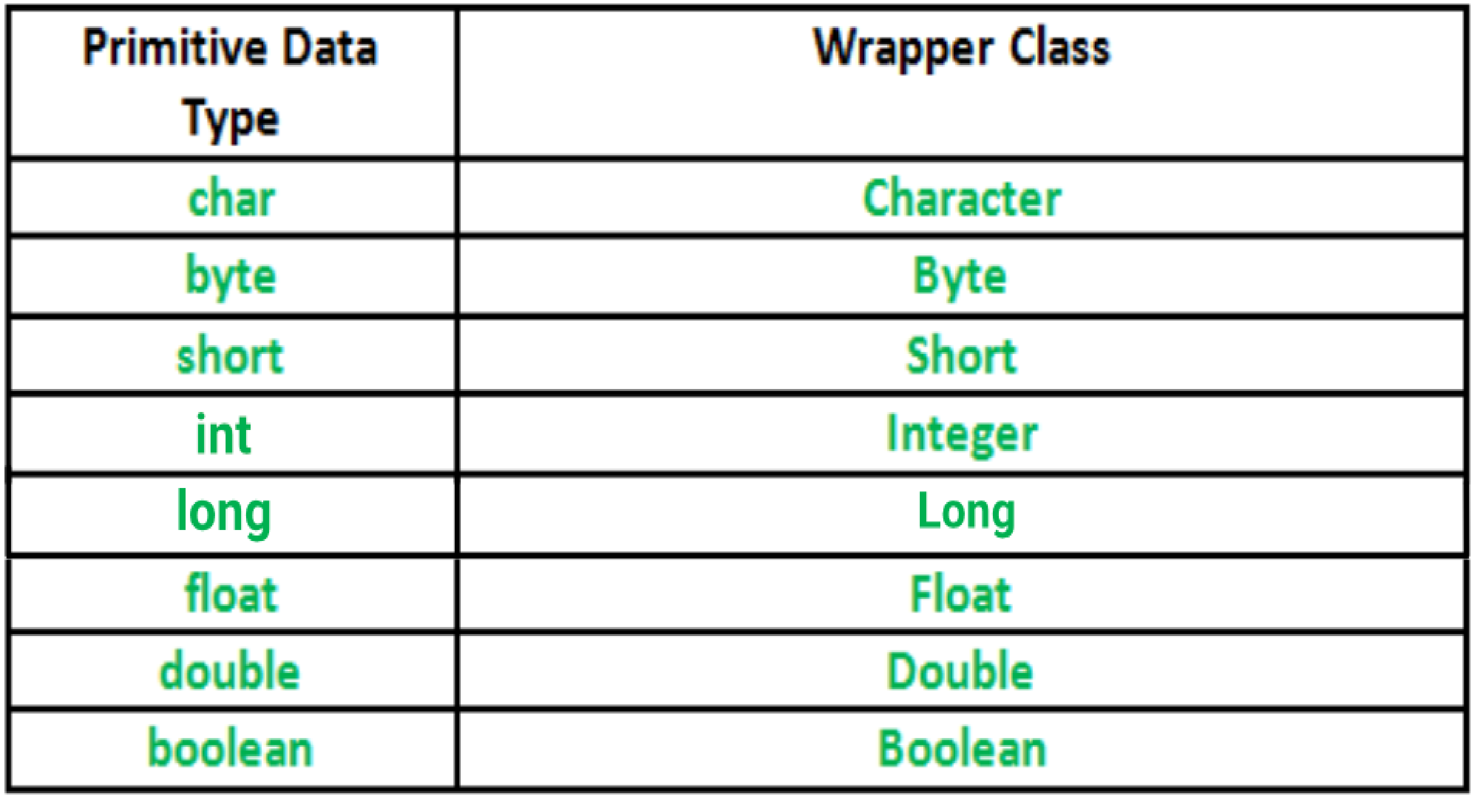
- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.

Wrapper Class

- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Wrapper Class



Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Wrapper Class

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Wrapper class Example: Primitive to Wrapper

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
System.out.println(a+" "+i+" "+j);
}}
```

Wrapper Class

Output- 20 20 20

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Wrapper class Example: Wrapper to Primitive

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}}
```

```
output-
3 3 3
```

Generics

Generics mean **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 called a generic entity.

Generics

The most commonly used generic type parameter names are:

- E – is supposed to be an element (widely used by Java Collections Framework)
- K – is supposed to be a key (in a Map < K, V >)
- N – is supposed to be a number
- T – is supposed to be a type
- V – is supposed to be a value (as a return value or a mapped value)
- S, U, V etc. – 2nd, 3rd, 4th types

Generics

Why Generics?

The **Object** is the superclass of all other classes and Object reference can refer to any type object. These features lack type safety. Generics add that type safety feature.

Generics

Generic Class

Like C++, we use <> to specify parameter types in generic class creation. To create objects of a generic class, we use the following syntax.

```
// To create an instance of generic class BaseType
 <Type> obj = new BaseType <Type>()
```

Note: `In Parameter type we can not use primitives like 'int','char' or 'double’.`

We can also pass multiple Type parameters in Generic classes.

Generics

Generic Functions:

We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to the generic method, the compiler handles each method.

Generics

Generics work only with Reference Types:

When we declare an instance of a generic type, the type argument passed to the type parameter must be a reference type. We cannot use primitive data types like **int**, **char.**

```
Test<int> obj = new Test<int>(20);
```

The above line results in a compile-time error, that can be resolved by using type wrappers to encapsulate a primitive type.
But primitive type array can be passed to the type parameter because arrays are reference type.

```
ArrayList<int[]> a = new ArrayList<>();
```

Generics

Advantages of Generics:

Programs that use Generics has got many benefits over non-generic code.

1. Code Reuse:

We can write a method/class/interface once and use it for any type we want.

2. Type Safety:

Generics make errors to appear compile time than at run time (It’s always better to know problems in your code at compile time rather than making your code fail at run time). Suppose you want to create an ArrayList that store name of students and if by mistake programmer adds an integer object instead of a string, the compiler allows it. But, when we retrieve this data from ArrayList, it causes problems at runtime.

Generics

Advantages of Generics:

3. Individual Type Casting is not needed: If we do not use generics, then, in the above example every time we retrieve data from ArrayList, we have to typecast it. Typecasting at every retrieval operation is a big headache. If we already know that our list only holds string data then we need not typecast it every time.

4. Generics promotes code reusability.

5. Implementing generic algorithms: By using generics, we can implement algorithms that work on different types of objects and at the same, they are type safe too.

_________ add stability to your code by making more of your bugs detectable at compile time.
Ans- Generics

Difference between Array and Array List

Q Compare Array and Array List with examples.

Ans- https://www.javatpoint.com/difference-between-array-and-arraylist

Daily Quiz

Q1. What is an event in delegation event model used by Java programming language?
a) An event is an object that describes a state change in a source.
b) An event is an object that describes a state change in processing.
c) An event is an object that describes any change by the user and system.
d) An event is a class used for defining object, to create events.

Q2. Which of these methods are used to register a keyboard event listener?

a) KeyListener()
b) addKistener()
c) addKeyListener()
d) eventKeyboardListener()

Daily Quiz

Q3.Which of these methods are used to register a mouse motion listener?

a) addMouse()

b) addMouseListener()

c) addMouseMotionListner()

d) eventMouseMotionListener()

Q4. Which statement is true about a static nested class?

A. You must have a reference to an instance of the enclosing class in order to instantiate it.

B. It does not have access to nonstatic members of the enclosing class.

C. It's variables and methods must be static.

D. It must extend the enclosing class.

Daily Quiz

Q5. Which of the following statements about static inner classes is true?

AA static inner class requires an instance of the enclosing class.

B.A static inner class requires a static initializer.

C.A static inner class has no reference to an instance of the enclosing class.

D.A static inner class has access to the non-static members of the outer class.

Q6. Which of these is not a interface in the Collections Framework?

a. Collection
b.Group
c.Set
d.List

Daily Quiz

Q7. Which of these collection class has the ability to grow dynamically?

a. Array
b. Arrays
c. ArrayList
d.(None of these)

Q8.Which is best suited to a multi-threaded environment?

a. WeakHashMap
b. Hashtable
c. HashMap
d. ConcurrentHashMap

Daily Quiz

Q9.Which of these object stores association between keys and values?
a) Hash table
b) Map
c) Array
d) String

Q10. What is a listener in context to event handling?

a) A listener is a variable that is notified when an event occurs.
b) A listener is a object that is notified when an event occurs.
c) A listener is a method that is notified when an event occurs.
d) None of the mentioned

Daily Quiz

Q11.The subclass of a java.awt.Component class is known as a ?

a. System

b. Component

c. Container

d. Component Manager

Q12. What is the super class of all components of Java?

A. java.all.Component

B. all.awt.Component

C. java.awt.Component

D. awt.Component

Daily Quiz

Q13. A ____________ dictates the style of arranging the components in a container.

A. Border

B. grid layout

C. panel

D. Layout Manager

Q14. What is the listener used to handle the events of a text field?

A. java.awt.ActionListener interface

B. java.awt.event.ActionListener

C. awt.event.ActionListener interface

D. java.awt.event.ActionListener interface

Daily Quiz

Q15.Where can the event handling code be written?

A. Same class

B. Other class

C. Anonymous class

D. All mentioned above

Q16.Which method is automatically called after the browser calls the init method?

A. Start

B. Stop

C. Destroy

D. Paint

Q17.Which of the following is true about primitives?

A. You can call methods on a primitive.

B. **You can convert a primitive to a wrapper class object simply by assigning it.**

C. You can convert a wrapper class object to a primitive by calling valueOf().

D. You can store a primitive directly into an ArrayList.

Daily Quiz

Q18. Which package is imported for swing components?

A. java.awt.*;

B. java.util.*;

C. javax.swing.*;

D. javax.awt.swing.*;

Q19. Which exception is thrown by the getConnection() method of the DriverManager class?

A. ConnectionNotFoundException

B. IOException

C. **SQLException**

D. ReadException

Q20. What does the following code do?

smt=con.createStatement();

A. A Prepared Statement object is created to send sql commands to the database

B. A Statement object is created to send sql commands to the database

C. A callable Statement is created to send sql commands to the database

D. A Statement object is created to execute parameterise SQL commands

Weekly Assignment

Q1.Analyze swing and what are the methods of component class in Java Swing?

Q2. What is a layout manager and what are different types of layout managers available in Java Swing?

Q3. Explain GUI with real life example.

Q4. Illustrate Container class with real life example.

Q5.Elaborate Array and List. Write a program for ArrayList.

Q6. Can you give an example of how Generics make a program more flexible?

Q7.Analyze Queues with real life example.

Q8.Discuss List, Set and Queue. How to synchronize List, Set and Map elements?

Q9. What are the various methods provided by the Queue interface?

Q10.Explain the features of Collection Framework in Java.

Q11.Construct CardLayout and describe.

Q12. Create Gridlayout and illustrate.

Q13. Elaborate Flowlayout with sketching.

Q14. Design Box Layout and discuss it.

Q16.Explain Border layout with sketching.

Topic Links

- https://www.youtube.com/watch?v=Ma7u6KEKzPE
- https://www.youtube.com/watch?v=2qWPpgALJyw
- https://www.youtube.com/watch?v=2qWPpgALJyw
- https://www.youtube.com/watch?v=tzihl-rBMJ0

MCQs

1. Which of the following is incorrect statement regarding the use of generics and parameterized types in Java?
A. Generics provide type safety by shifting more type checking responsibilities to the compiler
B. Generics and parameterized types eliminate the need for down casts when using Java Collections
C. When designing your own collections class (say, a linked list), generics and parameterized types allow you to achieve type safety with just a single class definition as opposed to defining multiple classes
Answer: C

2. Which of the following allows us to call generic methods as a normal method?
A. Type Interface
B. Interface
C. Inner class
D. All of the mentioned
Answer: A

MCQs

3. Why are generics used?
A. Generics make code more fast
B. Generics make code more optimised and readable
C. Generics add stability to your code by making more of your bugs detectable at compile time
D. Generics add stability to your code by making more of your bugs detectable at a runtime

Answer: C

4. Which of the following reference types cannot be generic?
A. Anonymous inner class
B. Interface
C. Inner class
D. All of the mentioned

Answer: A

MCQs

5. Which of these types cannot be used to initiate a generic type?
A. Integer class
B. Float Class
C. Primitive Types
D. Collections

Answer: C

6. Which of these instances cannot be created?
A. Integer Instance
B. Generic Class Instance
C. Generic Type Instance
D. Collection Instances

Answer: C

MCQs

7. Which of these is a correct way of defining of a generic method?
A. name(T1, T2, …, Tn) { /* … */ }
B. public name { /* … */ }
C. class name[T1, T2, …, Tn] { /* … */ }
D. name{T1, T2, …, Tn} { /* … */ }

Answer: B

8. Which of these package is used for graphical user interface?
java.applet
java.awt
java.awt.image
java.io

Answer: B

MCQs

9. Where are the following four methods commonly used?

1) public void add(Component c)
2) public void setSize(int width,int height)
3) public void setLayout(LayoutManager m)
4) public void setVisible(boolean)

a. Graphics class
b. Component class
c. Both A & B
d. None of the above
Answer: B

10. Which is the container that doesn't contain title bar and MenuBars but it can have other components like button, textfield etc?
a. Window
b. Frame
c. Panel
d. Container

Answer : C

MCQs

11. Which package provides many event classes and Listener interfaces for event handling?
a. java.awt
b. java.awt.Graphics
c. java.awt.event
d. None of the above

Answer: c

12. Which is a component in AWT that can contain another components like buttons, textfields, labels etc.?
a. Window
b. Container
c. Panel
d. Frame

Answer: b

MCQs

13. Which of these methods can be used to obtain the reference to the container that generated a ContainerEvent?

A. getContainer()

B. getContainerCommand()

C. getActionEvent()

D. **getContainerEvent()**

Q14. Which are passive controls that do not support any interaction with the user?

A. Choice

B. List

C. Labels

D. Checkbox

Q15. Which of these events generated when a button is pressed?

A. ActionEvent

B. WindowEvent

C. ItemEvent

D. KeyEvent

MCQs

Q16. ___________ arranges the components horizontally.

A. BorderLayout

B. CardLayout

C. GridLayout

D. FlowLayout

Q17. The Swing Component classes that are used in Encapsulates a mutually exclusive set of buttons?

A. AbstractButton

B. ButtonGroup

C. Jbutton

D. ImageIcon

Q18. A checkbox is a control that consists of a___________.

A. Combination of a small box

B. A label

C. Combination of a large box and a label

D. Both a & b

MCQs

Q19. ___________ are a set of Java class libraries provided as part of Java 2 Platform, Standard Edition (J2SE) to support building graphics user interface (GUI) and graphics functionality for client applications that will run on popular platforms such as Microsoft Windows, Linux, and Mac OSX.

A. Swing

B. Applet

C. AWT

D. Java Foundation Classes

Q20 .The classes and interfaces defined in AWT are contained within the _______ package.

A. java.awt.*;

B. java.sql.*;

C. java.io.*;

D. java.int*;

Thank You