Coding Bootcamp

Core Java

TECHNOLOGY

Java Collections

Learning Objectives

By the end of this lesson, you will be able to:

- Classify different Java class wrappers to improve data management and conversions
- Create and demonstrate applet programs to enhance web applications with interactive and dynamic content
- Analyze Graphic Class methods in applet programs to optimize client-side graphic rendering

Learning Objectives

By the end of this lesson, you will be able to:

- Evaluate applet programming's parameter usage and communication to boost functionality and ensure robust data interchange
- Develop and implement user interface programs with applet, utilizing collections like Vector, HashSet, TreeSet, and HashMap for better data structure management

TECHNOLOGY

Wrapper and Inner Class

Wrapper and Inner Class

Time period wrapper class refers to a specific group of Java classes that convert primitive Java types into objects.

Java has corresponding wrapper classes for each primitive type.

Wrapper classes encapsulate primitive types inside an object.

These wrapper classes are called type wrappers.

Example: The Integer class is the wrapper for the int primitive type.

Wrapper classes for converting simple primitive types:

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

Wrapper and Inner Class

The int or integer class:

Wraps the primitive type int value in an object

Includes a single field of type int

Provides methods for converting an int to a string and a string to an int object

Offers constants and additional methods for working with an int reference variable

Syntax:

```
public class StringToInteger {
  public static void main(String[] args) {

    //Convert String to Integer
    //1. Construct Integer by using a
constructor.
    Integer num1 = new Integer("110");
    System.out.println(num1);

    //2. Use the valueOf method of Integer
class.
    String str = "100";
    Integer num2 = Integer.valueOf(str);
    System.out.println(num1);
 }
}
```

Wrapper and Inner Class

The float class:

Wraps the float primitive type in an object

Includes a single field of type float

Provides methods for converting float values to string types and converting string objects to float objects

Offers constants and methods for working with float reference variables

Syntax:

```
/*
  Convert Float object to String object
  This example shows how a Float object
can be converted into a String object.
*/

public class JavaFloatToStringExample {

  public static void main(String[] args) {

    Float float_num = new Float(10.77);
    //use the toString method of Float
class to convert Float into String.
    String str_num = float_num.toString();
    System.out.println("Float converted to
String as " + str_num);
  }
}
```

Wrapper and Inner Class

A character class encapsulates a char value and provides a subsequent constructor.

Syntax:

```
character(char ch) //here,
c is a char value
```

The charValue() method returns the char value encapsulated by a character object.

Syntax:

```
char charValue()
```

Wrapper and Inner Class

Example

```
import Java.util.*;
public class Char_Demo
{
public static void main(String args(1)
 {
Character character = new Character()1;
char ch = '0'; System.out.println("Object ob = " +
character);
System.out.println("char value of Object = " +
character.charValue());
System.out.println(ch + " is defined character set ? " +
Character.isDefined(ch));
System.out.printIn("ob is lowercase character = " +
Character.isLowerCase(ch)); System.out.println("ob is
uppercase character = " + Character.isUpperCase(ch));
}
}
```

Wrapper and Inner Class

The Boolean class encapsulates a Boolean value. It defines **false** and **true** constants.

Class affords constructors:

```
Boolean(Boolean b) Boolean(String
str)
```

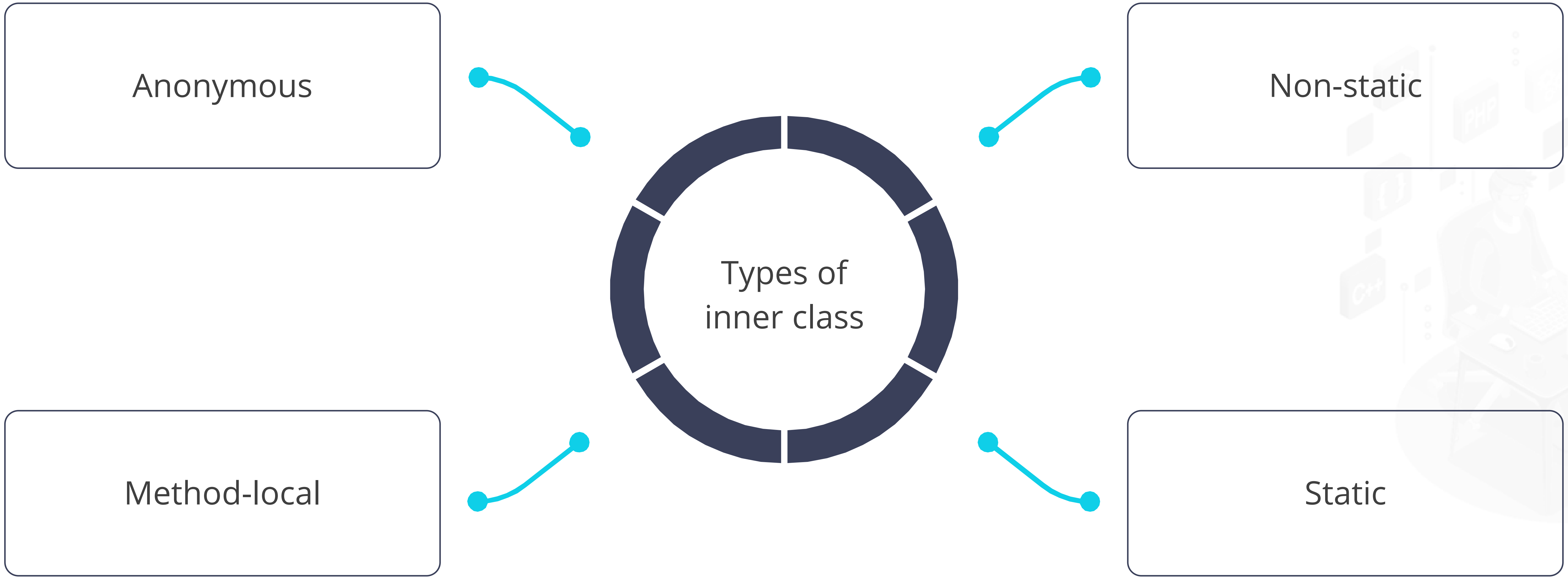
Here, b is a Boolean value, and str is the string equivalent of a Boolean value.

Syntax:

```
import Java.util*;
public class Boolean_check {
   public static void main(String args){
       Boolean b1 = new Boolcan(true);
       Boolean b2 = new Boolean(false);
       System.out.println("S1 =" + 55);
       Systems.out.println("S2 = " + b2);
       Boolean b3 = new Boolean("true");
       Boolean b4 = new Boolean("false");
       System.out.println("S3 = " + b3);
       System.out.println("ObJeet " +
55);
       System.out.println("Boolean Value
=" + b1.booleanValue();
System.out.printIn("(55==b2)? "+
b1.equals(b2));
   }
}
```

Wrapper and Inner Class

An inner class is a **nested class** declared inside a class or interface.



Implementing Wrapper Classes

Problem Statement:

You have been asked to implement Java wrapper classes to represent primitive data types as objects, enabling utility methods and use in collections.

Outcome:

By implementing Java wrapper classes to represent primitive data types as objects, you will learn to utilize utility methods and enable their use in collections. This knowledge is essential for leveraging the full capabilities of Java's standard library and handling primitive values in object-oriented contexts.

Note: Refer to the demo document for detailed steps: 01_Implementing_Wrapper_Classes

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a Java project
2. Create primitive variables
3. Add methods on the wrapper classes

Applet Programs

Applet Programs

An applet is a program designed to produce dynamic content embedded in a website. It operates on the client side within a browser.

The benefits of applet programs are:

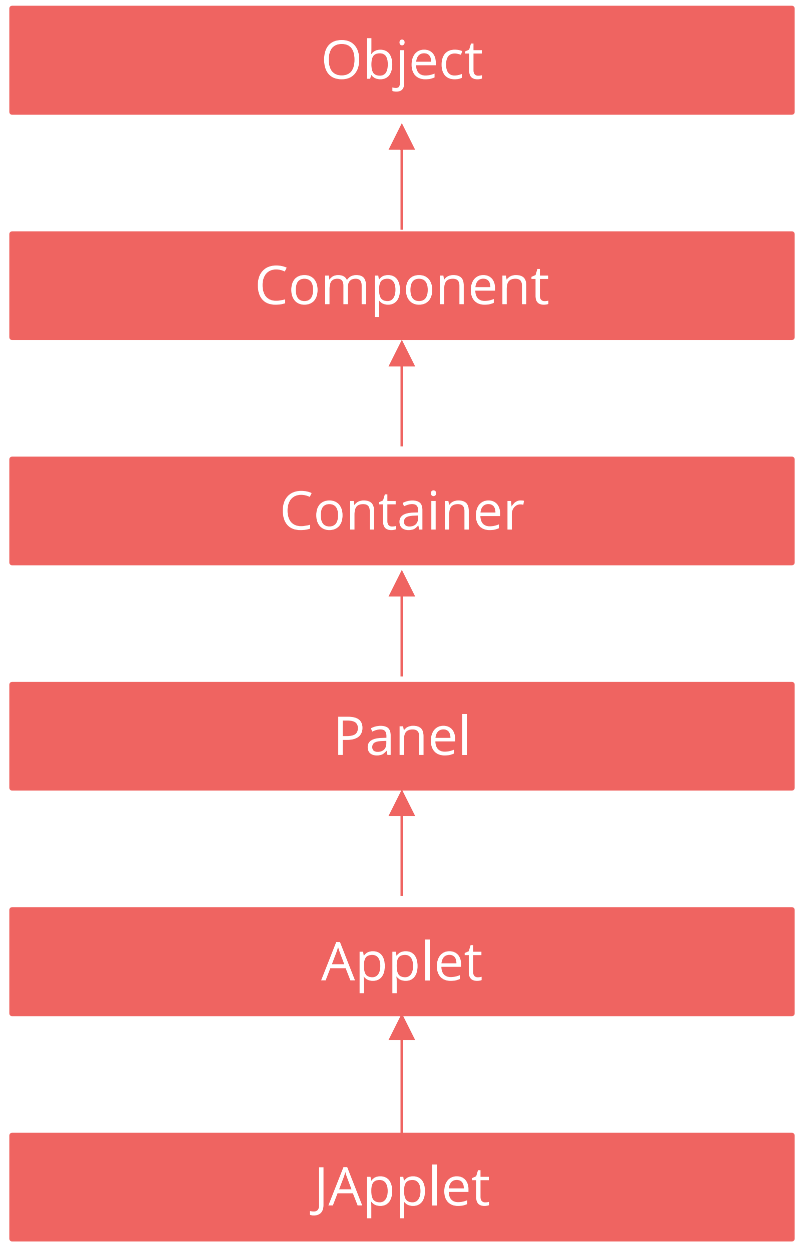
- ✓ It decreases response time for clients.
- ✓ It is highly secure.
- ✓ It can be executed on multiple platforms, including Linux, macOS, and Windows, through web browsers.

To run an applet, the client's browser requires a plugin.

Applet Hierarchy and Lifecycle

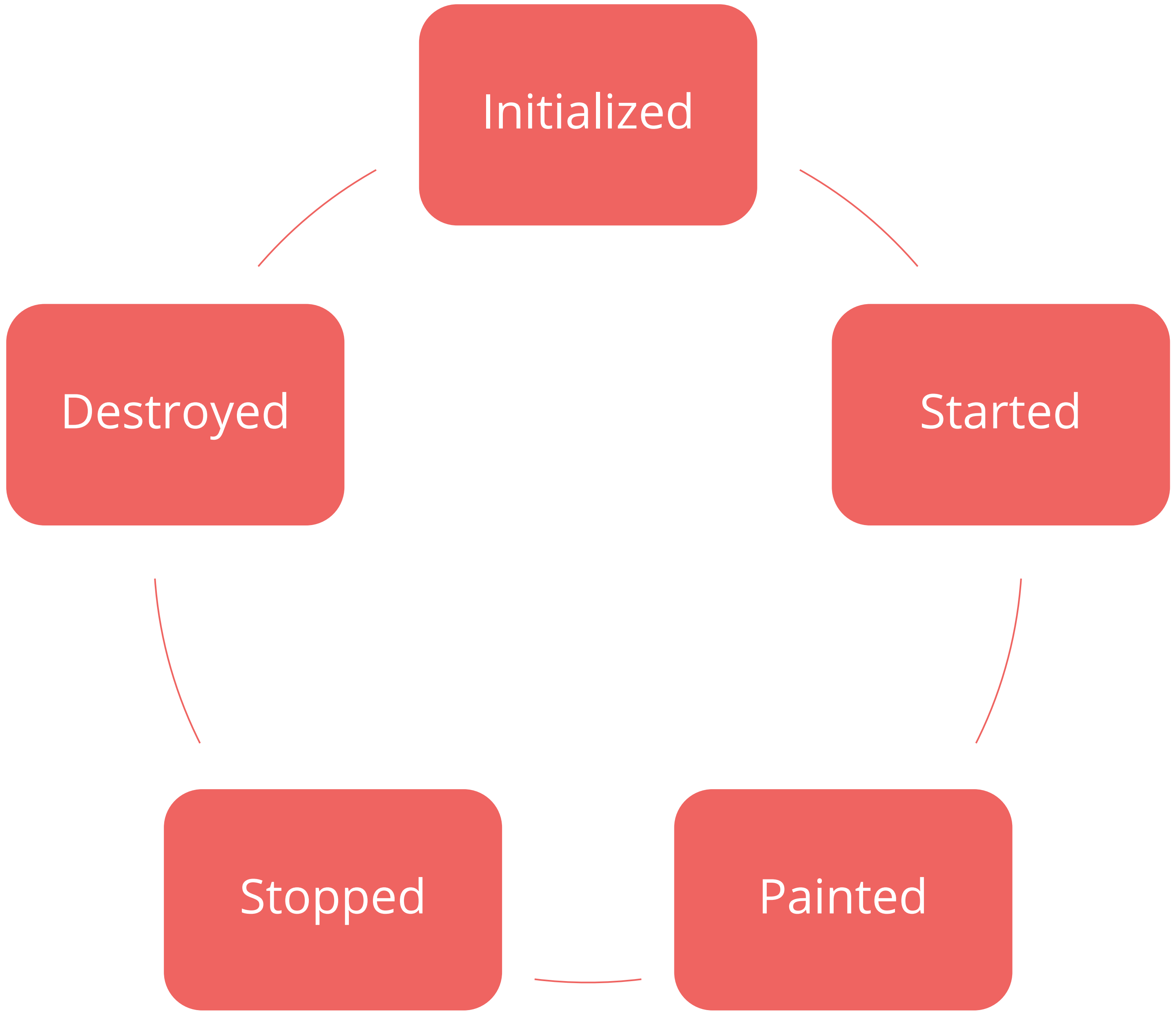
Applet Hierarchy

Hierarchy of the applet highlighting the superclass:



Applet Lifecycle

Lifecycle of applet:



Applet Lifecycle

Initialize the applet using the public void init()

Start the applet using the public void start()

Stop the applet using the public void stop()

Destroy the applet using public void ruin()

Applet Lifecycle

Create an applet with an applet tag in a comment and run it using the applet viewer tool.

Syntax

```
/*
<applet code="First.class"
width="260" height="260">
</applet>
*/
```

To run an applet using the applet viewer tool, use the following command:

```
D:\>Javac First.Java
D:\>appletviewer First.Java
```

Applet Lifecycle: Paint

Example

```
import Java.applet.Applet;
import Java.awt.Graphics;
public class First extends Applet{
    public void paint(Graphics g){
         g.drawString("Hello
World!!",260,260);
    }
}
```

Myapplet.html

```
<html>
<body>
    <applet code="First.class"
width="240" height="240">
    </applet>
</body>
</html>
```

Applet Lifecycle: Graphic Class

stringDraw() is used to attract the required string.

drawRect() is used to draw a rectangle with the required width.

fillRect() is used to fill a rectangle.

drawOval() is used to draw an oval.

fillOval() is used to fill an oval.

drawLine() is used to draw a line between the factors (x1, y1) and (x2, y2).

Syntax

```
public abstract void stringDraw(String str,
int a, int b)

public void drawRect(int a, int b, int width,
int top)

public abstract void fillRect(int a, int b,
int width, int peak)

public abstract void drawOval(int a, int b,
int width, int height)

public abstract void fillOval(int a, int b,
int width, int height)

public abstract void drawLine(int a1, int b1,
int a2, int b2)
```

Applet Lifecycle: Graphic Class

drawImage() is used to draw the specified image.

drawArc() is used to draw an elliptical or circular arc.

fillArc() is used to fill a round or elliptical arc.

setColor() is used to set photograph's shade to the required shade.

setFont() is used to set the photos.

Syntax

```
public abstract boolean drawImage(photograph
img, int a, int b, ImageObserver observer)

public abstract void drawArc(int a, int b,
int width, int peak, int startAngle, int
arcAngle)

public abstract void fillArc(int a, int b,
int width, int height, int startAngle, int
arcAngle)

public abstract void setColor(coloration c)

public abstract void setFont(Font font)
```

Applet Lifecycle: Graphic Class

Example

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

public class GraphicsExample extends
Applet{

public void paint(Graphics g){
g.setColor(Color.red);
g.stringDraw("Hello!!",60, 60);
g.drawLine(30,40,30,300);
g.drawRect(60,100,20,20);
g.fillRect(160,100,20,20);
g.drawOval(60,200,20,20);

g.setColor(Color.yellow);
g.fillOval(160,200,20,20);
g.drawArc(80,150,20,20,20,270);
g.fillArc(270,140,20,20,0,170);
}
}
```

Myapplet.html

```
<html>
<body>
<applet code="GraphicsExample.class"
width="230" height="230">
</applet>
</body>
</html>
```

Applet Lifecycle: Applet Class

The applet class methods that are necessary for displaying a photo are:

getDocumentBase()

The first method is public URL **getDocumentBase()**, which returns the URL of the webpage where the Applet is currently embedded.

getCodeBase()

The second method is public URL **getCodeBase()**, which provides the base URL that can be used to resolve relative URLs for the applet's resources, such as image files.

Applet Lifecycle: Applet Class

Example of displaying images in applet

```
import Java.awt.*;
import Java.applet.*;

public class DisplayImage extends Applet
{
  Image picture;
  public void init() {
    picture =
getImage(getDocumentBase(),"hello1.jpg")
;
  }

  public void paint(Graphics g) {
    g.drawImage(picture, 35,35, this);
  }
  }
```

Here, the drawImage() approach of the graphics class is used to display the photo.

Syntax

```
import Java.awt.*;
import Java.applet.*;
public class AnimationExample extends Applet {
  Image picture;
  public void init() {
    picture
=getImage(getDocumentBase(),"hello.gif");
  }
  public void paint(Graphics g) {
    for(int i=0;i<500;i++){
      g.drawImage(picture, i,30, this);
      try{Thread.sleep(100);}catch(Exception e){}
    }
  }  }
```

TECHNOLOGY

Event Handling in Applet

EventHandling in Applet

The process of handling events in an applet is similar to that of handling events in AWT or Swing.

Example:

```
import Java.applet.*;
import Java.awt.*;
import Java.awt.event.*;
public class EventApplet extends Applet implements
ActionListener{
Button b;
TextField tf;
public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);
b=new Button("Click");
b.setBounds(80,150,60,50);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
 public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }   }
```

Here, users have created all of the controls in the init() method.

TECHNOLOGY

JApplet Class in Applet

JApplet Class in Applet

If users choose to switch from AWT to Swing, they can utilize the components of Swing within a JApplet.

JApplet class extending the Applet class

```
import Java.applet.*;
import Javax.swing.*;
import Java.awt.event.*;
public class EventJApplet extends JApplet implements
ActionListener{
JButton b;
JTextField tf;
public void init(){
tf=new JTextField();
tf.setBounds(30,40,150,20);
b=new JButton("Click");
b.setBounds(80,150,70,40);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);   }
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}  }
```

Myapplet.html

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

Here, all of the controls are created in the init() method.

TECHNOLOGY

Painting in Applet

simplilearn

Painting in Applet

The mouseDragged() method of the MouseMotionListener is used to carry out painting tasks within an applet.

Syntax:

```
import Java.awt.*;
import Java.awt.event.*;
import Java.applet.*;
public class MouseDrag extends Applet
implements MouseMotionListener{
public void init(){
addMouseMotionListener(this);
setBackground(Color.green);   }
public void mouseDragged(MouseEvent me){
Graphics g=getGraphics();
g.setColor(Color.yellow);
g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me){}
}
```

Myapplet.html

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

MouseEvent's getX() and getY() methods retrieve the current x and y coordinates. The getGraphics() method from the factor class returns a snapshot object.

TECHNOLOGY

Parameter in Applet

Parameter in Applet

The getParameter() method within the applet class allows for the extraction of any information included in the HTML report as a parameter.

Syntax:

```
public String getParameter(String parameterName)
```

Example of using parameters in applet:

```
import Java.applet.Applet;
import Java.awt.Graphics;

public class UseParam extends
Applet{
public void paint(Graphics g){
String str=getParameter("msg");
g.drawString(str,50, 50);
}
}
```

Myapplet.html

```
<html>
<body>
<applet code="UseParam.class"
width="300" height="300">
<param name="msg" value="Welcome
to applet">
</applet>
</body>
</html>
```

TECHNOLOGY

Applet Communication

simplilearn

Applet Communication

Java.applet.AppletContext facilitates applet communication and allows applet renaming via HTML. To retrieve the applet object, it contains the getApplet() method.

Syntax

```
public Applet getApplet(String name){}
```

Example:

```
import Java.applet.*;
import Java.awt.*;
import Java.awt.event.*;
public class ContextApplet extends Applet implements
ActionListener{
Button b;
public void init(){
b=new Button("Click");
b.setBounds(50,50,60,50); add(b);
b.addActionListener(this);  }
public void actionPerformed(ActionEvent e){
AppletContext ctx=getAppletContext();
Applet a=ctx.getApplet("app2");
a.setBackground(Color.yellow);  }  }
```

Myapplet.html

```
<html>
<body>
<applet code="ContextApplet.class"
width="150" height="150" name="app1">
</applet>
<applet code="First.class" width="150"
height="150" name="app2"> </applet>
</body>
</html>
```

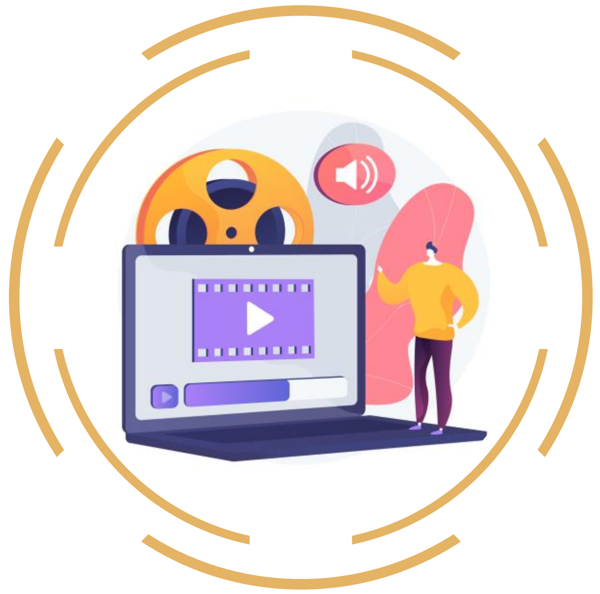
TECHNOLOGY

Writing User Interface Program with Applet

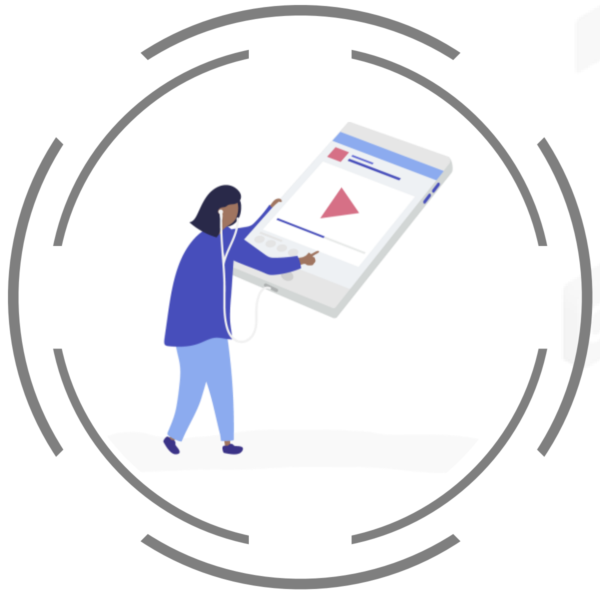
Writing User Interface Program with Applet

The applet class offers essential functionalities for executing applets, including the ability to initialize and terminate them.

Display audio



Videos



Playback images

Writing User Interface Program with Applet

Methods that applets override:

init()	This method initializes the reference variable and is called only once during applet execution.
start()	This method is called after init() and restarts the applet when stopped.
stop()	This method dismisses threads that are not needed when the applet is not visible.
destroy()	This method removes the applet from memory.

Syntax:

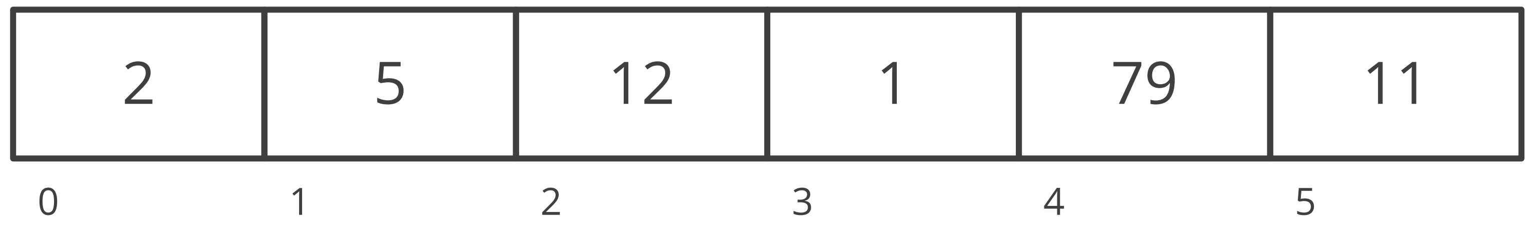
```
import Java.awt.*;
import Java.applet.*;
public class AppletDemo extends Applet {
    public void init() {
//initialization of reference variable
}
    public void start () { //start or
resume execution  }
    public void stop() { //suspend
execution of applet}
    public void destroy() { //perform
shutdown operation of applet  }
    public void paint (Graphics g) {
//display the content on the applet
window  }
}
```

ArrayList

ArrayList

ArrayList is a useful collection framework that belongs to Java.util package.

ArrayList Integer Object Type:



Integer type (for all indices) Data

This framework allows dynamic arrays to be created in Java, which is useful for various applications that require array manipulation.

ArrayList

Example:

```
import Java.io.*;
import Java.util.*;
class ArrayList {
    public static void main(String[] args){
        // Size of the ArrayList
        int n = 6;
      // Declaring the ArrayList with initial size n
        ArrayList<Integer> arrlist
            = new ArrayList<Integer>(n);
// Appending new elements at the end of the list
        for (int i = 1; i <= n; i++)
            arrlist.add(i);
        // Printing elements
        System.out.println(arrli);
        // Remove element at index 2
        arrlist.remove(2);
        // Displaying the ArrayList after deletion
        System.out.println(arrlist);
        // Printing elements one by one
        for (int i = 0; i < arrlist.size(); i++)
            System.out.print(arrlist.get(i) + " ");
    }  }
```

Output:

```
[1, 2, 3, 4, 5,6]
[1, 2, 4, 5, 6]
1 2 4 5 6
```

ArrayList

ArrayLists are dynamic arrays, so users do not need to specify their size. As they include or exclude elements, their capacity automatically grows.

The following process takes place when an array reaches its maximum capacity:

A new memory with a larger size is allocated to the heap memory.

The existing memory elements in the array are copied to the newly allocated memory.

The array now points to newly allocated memory with a larger size.

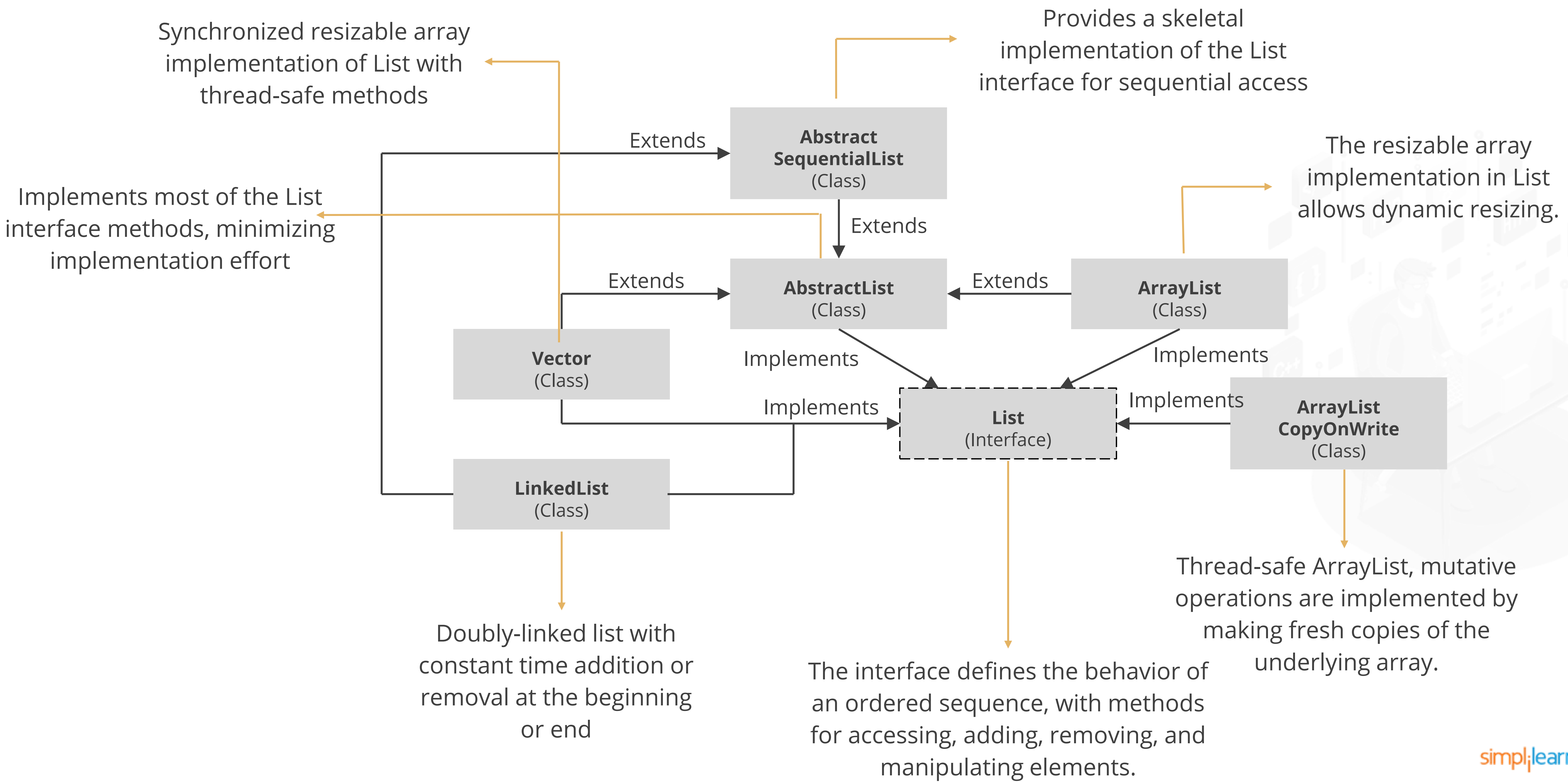
The old memory that was previously allocated for the array is deleted or freed up.

ArrayList

Essential functions of an arraylist:

- Is not synchronized
- Can be visible as a vector in C++
- Cannot be used for primitive sorts
- Inherits the abstractList class and implements the list interface
- Initializes with the aid of the scale
- Permits users to randomly get right of entry to the list

ArrayList: Hierarchy Diagram


Synchronized resizable array implementation of List with thread-safe methods
Provides a skeletal implementation of the List interface for sequential access
Extends
Abstract SequentialList (Class)
Extends
The resizable array implementation in List allows dynamic resizing.
Implements most of the List interface methods, minimizing implementation effort
Extends
AbstractList (Class)
Extends
ArrayList (Class)
Vector (Class)
Implements
Implements
Implements
List (Interface)
Implements
ArrayList CopyOnWrite (Class)
LinkedList (Class)
Doubly-linked list with constant time addition or removal at the beginning or end
The interface defines the behavior of an ordered sequence, with methods for accessing, adding, removing, and manipulating elements.
Thread-safe ArrayList, mutative operations are implemented by making fresh copies of the underlying array.

Creating ArrayList

Problem Statement:

You have been asked to implement the use of ArrayList in Java for managing the shopping cart functionality in an e-commerce application.

Outcome:

By implementing the use of ArrayList in Java for managing the shopping cart functionality in an e-commerce application, you will learn to dynamically store and manipulate a collection of items. This enhances your ability to manage varying quantities of products efficiently, ensuring a smooth user experience.

Note: Refer to the demo document for detailed steps: 02_Creating_ArrayList

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Open Eclipse IDE and create a new class
2. Add code for shopping cart and run the project

TECHNOLOGY

Vector

Vector

Vector refers to the legacy lesson class that implements a scalable array of objects.

It supports the List interface, allowing the use of its methods.

Vector enables the implementation of dynamic arrays with integer index access.

It is synchronized and has a legacy approach similar to the ArrayList.

Vector's iterators are fail-fast, throwing a ConcurrentModificationException during concurrent modification.

Syntax

```
public class Vector<E> extends
AbstractList<E> implements
List<E>, RandomAccess, Cloneable,
Serializable
```

Here, E is a class that implements the list interfaces and extends AbstractList. Additionally, it directly acknowledges its subclasses.

Vector

Different constructors in the Vector class:

Vector()

It helps to create the default Vector of the initial capacity of 10, as shown below:

```
Vector<E> vector = new
Vector<E>();
```

Vector(int size)

It helps to create the Vector, whose initial capacity is characterized by its size, as shown below:

```
Vector<E> vector = new
Vector<E>(int size);
```

Vector(int length, int incr)

It specifies a list of factors to signify when the Vector is resized upward, as shown below:

```
Vector<E> vector = new
Vector<E>(int size, int
incr);
```

Vector

Example of creating and using a vector

```
import Java.io.*;
import Java.util.*;
class VectorExample {
    public static void main(String[] args){
        // Size of the Vector
        int n = 5;
        // Declaring the Vector with initial size n
Vector<Integer> vector = new Vector<Integer>(n);
// Appending new elements at the end of the vector
        for (int i = 1; i <= n; i++)
            vector.add(i);
        // Printing elements
        System.out.println(vector);
        // Remove element at index 3
        vector.remove(3);
        // Displaying the vector after deletion
        System.out.println(vector);
        // Printing elements one by one
        for (int i = 0; i < vector.size(); i++)
            System.out.print(vector.get(i) + " ");
    }  }
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Vector

The upload() technique is used to upload the elements to the **Vector**.

It performs operations based on distinctive parameters. They are:

add(item) is used to add detail to the cease of the Vector.

add(int index, item) is used to feature detail at a specific index in the Vector.

Syntax

```
import Java.util.*;
import Java.io.*;
class AddElementsToVector {
    public static void main(String[] arg) {
        // create default vector
        Vector v1 = new Vector();
// Add elements using add() method
        v1.add(1);
        v1.add(2);
        v1.add("Hello");
        v1.add("World");
        v1.add(3);
          // print the vector to the console
        System.out.println("Vector v1 is " + v1);
        // create generic vector
        Vector<Integer> v2 = new
Vector<Integer>();
        v2.add(1);
        v2.add(2);
        v2.add(3);
        System.out.println("Vector v2 is " + v2);
    }  }
```

Vector

Use the set() method and provide the index of the element to be replaced with the new value to modify a vector element.

Syntax

```
import Java.util.*;
public class UpdatingVector {
    public static void main(String args[]) {
// Creating an empty Vector
Vector<Integer> vector = new Vector<Integer>();
// Use add() method to add elements in the vector
        vector.add(12);
        vector.add(23);
        vector.add(22);
        vector.add(10);
        vector.add(20);
// Displaying the Vector
        System.out.println("Vector: " + vector);
// Using set() method to replace 12 with 22
System.out.println("The Object that is replaced is:
"+ vector.set(0, 21));
// Using set() method to replace 20 with 60
System.out.println("The Object that is replaced is:
"+ vector.set(4, 50));
// Displaying the modified vector
        System.out.println("The new Vector is:" +
vector); } }
```

Output

```
Vector: [12, 23, 22, 10, 20]
The Object that is replaced is: 12
The Object that is replaced is: 20
The new Vector is:[22, 23, 22, 10, 60]
```

Vector

The dispose of() method can eliminate specific details from a vector.

Syntax

```
import Java.util.*;
import Java.io.*;
class RemovingElementsFromVector {
    public static void main(String[] arg)
    {
// create default vector of capacity 10
        Vector v = new Vector();
// Add elements using add() method
        v.add(1);
        v.add(2);
        v.add("Hello");
        v.add("Hi");
        v.add(4);
// removing first occurrence element at 1
        v.remove(1);
// checking vector
        System.out.println("after removal: " + v);
    }  }
```

This method has multiple overloads, allowing different operations to be performed depending on the parameters passed.

Vector

Various methods can be used to iterate through a vector, the most common being a for loop in combination with the get() method.

Syntax

```
import Java.util.*;
public class IteratingVector {
    public static void main(String args[])
    {
          // create an instance of vector
        Vector<String> vector = new Vector<>();
          // Add elements using add() method
        vector.add("Hello");
        vector.add("HI");
        vector.add(1, "For");
        // Using the Get method and the
        // for loop
        for (int i = 0; i < vector.size(); i++) {
            System.out.print(vector.get(i) + " ");
        }
        System.out.println();
        // Using the for each loop
        for (String str : vector)
            System.out.print(str + " ");
    }}
```

TECHNOLOGY

HashSet

HashSet

It is a set implementation that doesn't maintain element order and allows null values. It offers efficient CRUD operations, assuming proper hash function distribution.

Implements a set interface

Removes duplicate values

Does not guarantee the inserted items in HashSet will be inserted in the same order

Allows NULL elements

Implements serializable and cloneable interfaces

HashSet

Extends the abstract set class and implements set, cloneable, and serializable interfaces

Sets the initial ability to excessive if the new release's performance is crucial

LinkedHashSet

Explores the number of buckets whilst a hashtable is created

The variety of buckets will be mechanically improved if the cutting-edge size gets full.

HashSet

The load factor measures how full the HashSet can get earlier than its capability is robotically multiplied.

Formula for load factor

```
                   Number of stored elements in the table
Load Factor =    -----------------------------------------
                   Size of the hash table
```

HashSet

Operations on HashSet:

Syntax to add elements

```
import Java.util.*;
import Java.io.*;
class AddingElementsToHashSet {
public static void main(String[] args)
    {
        // Instantiate an object
        // of HashSet
        HashSet<String> hs = new HashSet<String>();
// Elements are added using add() method
        hs.add("hi");
        hs.add("hello");
        hs.add("How are you");
// Print the contents on the console
System.out.println("HashSet elements : " + hs);
    }
}
```

HashSet

Syntax to remove the elements

```
import Java.io.*;
import Java.util.*;
class RemoveElementsOfHashSet {
public static void main(String[] args) {
        // Instantiate an object of HashSet
        HashSet<String> hs = new HashSet<String>();
        // Elements are added using add() method
        hs.add("hi");
        hs.add("hello");
        hs.add("how are you");
        hs.add("A");
        hs.add("B");
        hs.add("Z");
        // Print the contents on the console
        System.out.println("Initial HashSet " + hs);
        // Removing the element B
        hs.remove("B");
        // Print the contents on the console
System.out.println("After removing element " + hs);
// Returns false if the element is not present
System.out.println("Element AC exists in the Set : " + hs.remove("AC"));
    }}
```

HashSet

Syntax for iterating through the HashSet

```
import Java.io.*;
import Java.util.*;
class IterateTheHashSet {
public static void main(String[] args){
// Instantiate an object of HashSet
        HashSet<String> hs = new HashSet<String>();
        // Elements are added using add() method
        hs.add("hi");
        hs.add("hello");
        hs.add("how are you");
        hs.add("A");
        hs.add("B");
        hs.add("Z");
        // Iterating though the HashSet
        Iterator itr = hs.iterator();
        while (itr.hasNext())
System.out.print(itr.next() + ", ");
        System.out.println();
        // Using enhanced for loop
        for (String s : hs)
            System.out.print(s + ", ");
        System.out.println();
    } }
```

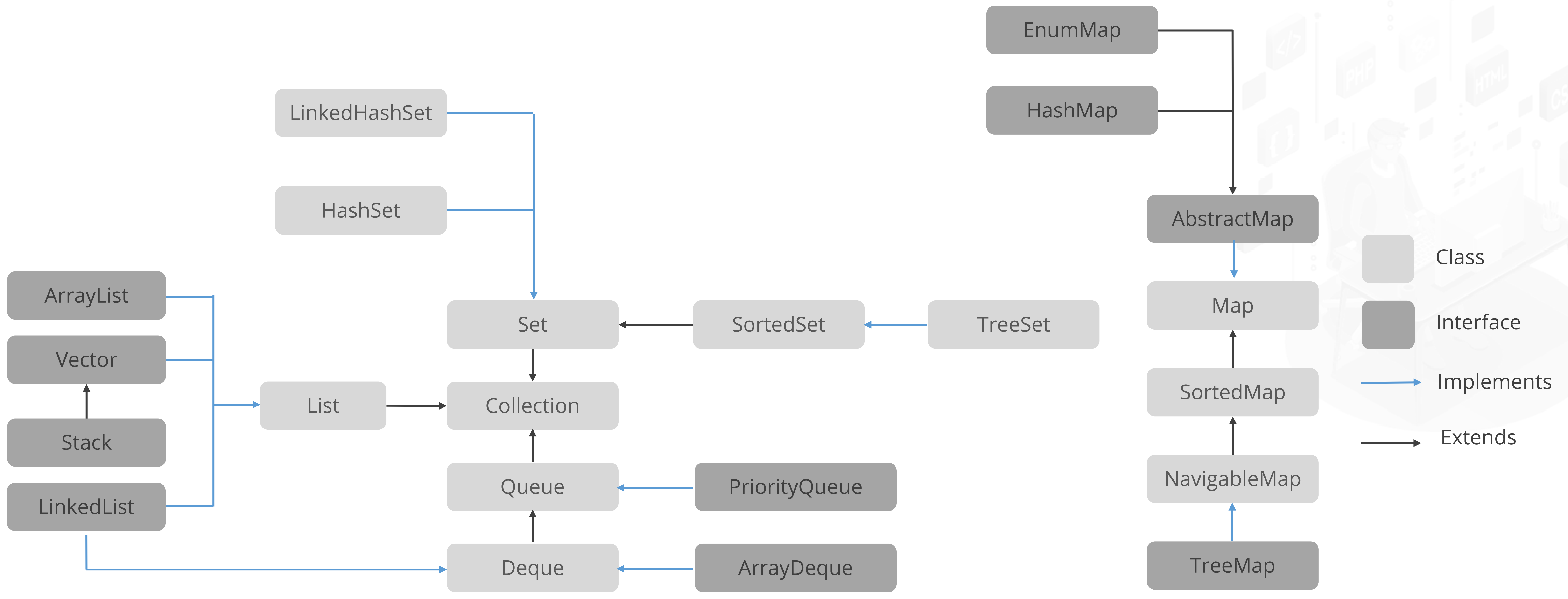
TECHNOLOGY

TreeSet

TreeSet

It is a Java implementation of SortedSet. Using a tree for storage and maintaining element order based on their natural ordering, which must be consistent with equals

A navigable set extends the sorted set interface:



TreeSet

If a set does not hold the insertion order:

A navigable set interface starts the implementation to explore the set.

Class TreeSet starts referring to a self-balancing tree.

Interface shows the way to navigate via this tree.

TreeSet

Operations performed in TreeSet:

Syntax to add elements

```
// Java code to Illustrate Addition of Elements to TreeSet
// Importing utility classes
import Java.util.*;
// Main class
class hihello{
      // Main driver method
      public static void main(String[] args)
      {
// Creating a Set interface with reference to TreeSet class Declaring
object of string type
             Set<String> ts = new TreeSet<>();
// Elements are added using add() method
             ts.add("hi");
             ts.add("hello");
             ts.add("how are you");
// Print all elements inside object
             System.out.println(ts);
      }}
```

TreeSet

Syntax to access the elements

```
// Java code Illustrate Working of TreeSet by
// Accessing the Element of TreeSet
// Importing utility classes
import Java.util.*;
// Main class
class hihello{
// Main driver method
public static void main(String[] args) {
// Creating a NavigableSet object with reference to
TreeSet class
NavigableSet<String> ts = new TreeSet<>();
// Elements are added using add() method
ts.add("hi");
ts.add("hello");
ts.add("how are you");
// Printing the elements inside the TreeSet object
System.out.println("Tree Set is " + ts);
String check = "hi";
// Check if the above string exists in
// the treeset or not
System.out.println("Contains " + check + " ‶ +
ts.contains(check));
```

Syntax to access the elements

```
// Print the first element in
// the TreeSet
System.out.println("First Value " + ts.first());
 // Print the last element in
// the TreeSet
System.out.println("Last Value " + ts.last());
String val = "hello";
// Find the values just greater
// and smaller than the above string
System.out.println("Higher " + ts.higher(val));
System.out.println("Lower " + ts.lower(val));
}}
```

TreeSet

Syntax to remove the values

```
// Java Program to Illustrate Removal of Elements in a
TreeSet
// Importing utility classes
import Java.util.*;
// Main class
class hihello{
// Main driver method
public static void main(String[] args)
{
// Creating an object of NavigableSet with reference to
TreeSet class
// Declaring object of string type
NavigableSet<String> ts = new TreeSet<>();
// Elements are added using add() method
ts.add("hi");
ts.add("hello");
ts.add("how are you");
ts.add("A");
ts.add("B");
ts.add("Z");
// Print and display initial elements of TreeSet
System.out.println("Initial TreeSet " + ts);
// Removing a specific existing element inserted above
ts.remove("B");
```

Syntax to remove the values

```
              // Printing the updated TreeSet
              System.out.println("After
removing element " + ts);
              // Now removing the first
element using pollFirst() method
              ts.pollFirst();
              // Again printing the updated
TreeSet
              System.out.println("After
removing first " + ts);
              // Removing the last element
using pollLast() method
              ts.pollLast();
              // Lastly printing the elements
of TreeSet remaining to figure out pollLast()
method
              System.out.println("After
removing last " + ts);
       }}
```

TreeSet

Syntax to iterate through the TreeSet

```
// Java Code for Illustrating the Working of TreeSet
// Importing utility classes
import Java.util.*;
// Main class
class GFG {
      // Main driver method
      public static void main(String[] args)  {
// Creating an object of Set with reference to TreeSet class
            Set<String> ts = new TreeSet<>();
      // Adding elements in above object using add() method
            ts.add("Hello");
            ts.add("world");
            ts.add("how are you");
            ts.add("B");
            ts.add("c");
            ts.add("x");
// Now with the helo of for each loop in order to iterate through the TreeSet
            for (String value : ts)
            // Printing the values inside the object
            System.out.print(value + ", ");
            System.out.println();
      } }
```

Differentiating Set and List

Problem Statement:

You have been asked to demonstrate the difference between Set and List in Java, highlighting their unique features and usage scenarios for managing email addresses, focusing on handling duplicates, and maintaining order.

Outcome:

By demonstrating the difference between Set and List in Java, you will learn that Set prevents duplicates, making it ideal for managing unique email addresses, while List maintains order, allowing for the tracking of email entry sequences. Understanding these distinctions helps in choosing the appropriate collection type based on the specific requirements of handling duplicates and maintaining order.

Note: Refer to the demo document for detailed steps: 03_Differentiating_Set_and_List

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new project
2. Create an ArrayList of type String to store email addresses
3. Handle duplicate records using the ArrayList
4. Execute the code with example data
5. Implement data storage using HashSet
6. Clear the emails using the clear method
7. Iterate over the email addresses using an iterator
8. Implement the LinkedHashSet and execute the code with example data

Implementing PriorityQueue

Problem Statement:

You have been asked to demonstrate the use of a priority queue in Java for sorting and processing data with a first-in-first-out approach.

Outcome:

By demonstrating the use of a priority queue in Java, you will learn how to sort and process data based on priority rather than a first-in-first-out approach. This allows you to manage tasks or data elements more efficiently, ensuring that higher-priority items are handled before others.

Note: Refer to the demo document for detailed steps: 04_Implementing_PriorityQueue

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new project
2. Add data to the queue using the add method
3. Implement the use of peek and poll methods
4. Implement the use of the variable size using the queue.size() method
5. Use the queue.iterator() function with example data
6. Implement queue prioritization

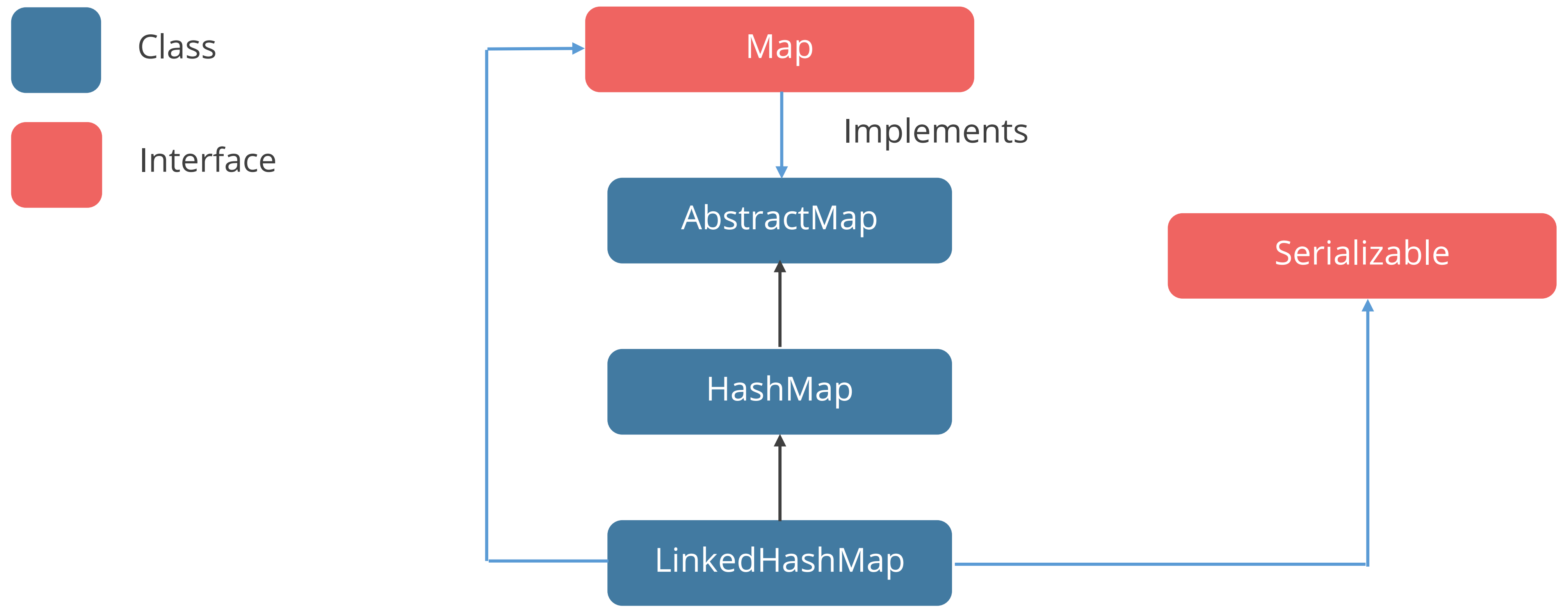
TECHNOLOGY

HashMap

HashMap

It is part of Java's collection located in Java.util package deal.

Stores the information



HashMap can be thought of as a HashTable, however:

It is unsynchronized.

It helps to store the null keys.

It must have at least one null key.

HashMap

Import Java.util.HashMap or its superclass to use this class and its strategies, as the map order is not guaranteed

Syntax:

```
HashMap<K, V> hm = new HashMap<K,
V>();
```

Example:

```
// Java program to Demonstrate the HashMap() constructor
import Java.io.*;
import Java.util.*;
// Main class
// To add elements to HashMap
class GFG {
      // Main driver method
      public static void main(String args[]) {
            // No need to mention the Generic type twice
            HashMap<Integer, String> hm1 = new HashMap<>();
      // Initialization of a HashMap using Generics
HashMap<Integer, String> hm2= new HashMap<Integer, String>();
// Adding elements using put method Custom input elements
            hm1.put(1, "one");
            hm1.put(2, "two");
            hm1.put(3, "three");
            hm2.put(4, "four");
            hm2.put(5, "five");
            hm2.put(6, "six");
            // Print and display mapping of HashMap 1
System.out.println("Mappings of HashMap hm1 are : "+ hm1);
            // Print and display mapping of HashMap 2
System.out.println("Mapping of HashMap hm2 are : "+ hm2); } }
```

HashMap

Operations on HasMap:

Syntax to add elements

```
// Java program to add elements to the HashMap
import Java.io.*;
import Java.util.*;
class AddElementsToHashMap {
      public static void main(String args[]) {
// No need to mention the Generic type twice
HashMap<Integer, String> hm1 = new HashMap<>();
// Initialization of a HashMap using Generics
HashMap<Integer, String> hm2
= new HashMap<Integer, String>();
// Add Elements using put method
             hm1.put(1, "hi");
             hm1.put(2, "hello");
             hm1.put(3, "how are you");
             hm2.put(1, "hi");
             hm2.put(2, "hello");
             hm2.put(3, "how are you");
System.out.println("Mappings of HashMap hm1 are : "+
hm1);
System.out.println("Mapping of HashMap hm2 are : "+ hm2);
      } }
```

Syntax to change elements

```
// Java program to change elements of
HashMap
import Java.io.*;
import Java.util.*;
class ChangeElementsOfHashMap {
public static void main(String args[])
{
// Initialization of a HashMap
HashMap<Integer, String> hm
= new HashMap<Integer, String>();
// Change Value using put method
             hm.put(1, "hi");
             hm.put(2, "hi");
             hm.put(3, "hi");
System.out.println("Initial Map " + hm);
hm.put(2, "For");
System.out.println("Updated Map " + hm);
}
}
```

HashMap

Syntax to remove elements

```
// Java program to remove elements from HashMap
import Java.io.*;
import Java.util.*;
class RemoveElementsOfHashMap{
      public static void main(String args[])
      {
            // Initialization of a HashMap
            Map<Integer, String> hm
                  = new HashMap<Integer, String>();
            // Add elements using put method
            hm.put(1, "hi");
            hm.put(2, "hello");
            hm.put(3, "ok");
            hm.put(4, "bye");
            // Initial HashMap
System.out.println("Mappings of HashMap are : "+ hm);
            // remove element with a key using remove method
            hm.remove(4);
            // Final HashMap
System.out.println("Mappings after removal are : "+ hm);
      } }
```

TECHNOLOGY

HashTable

simplilearn

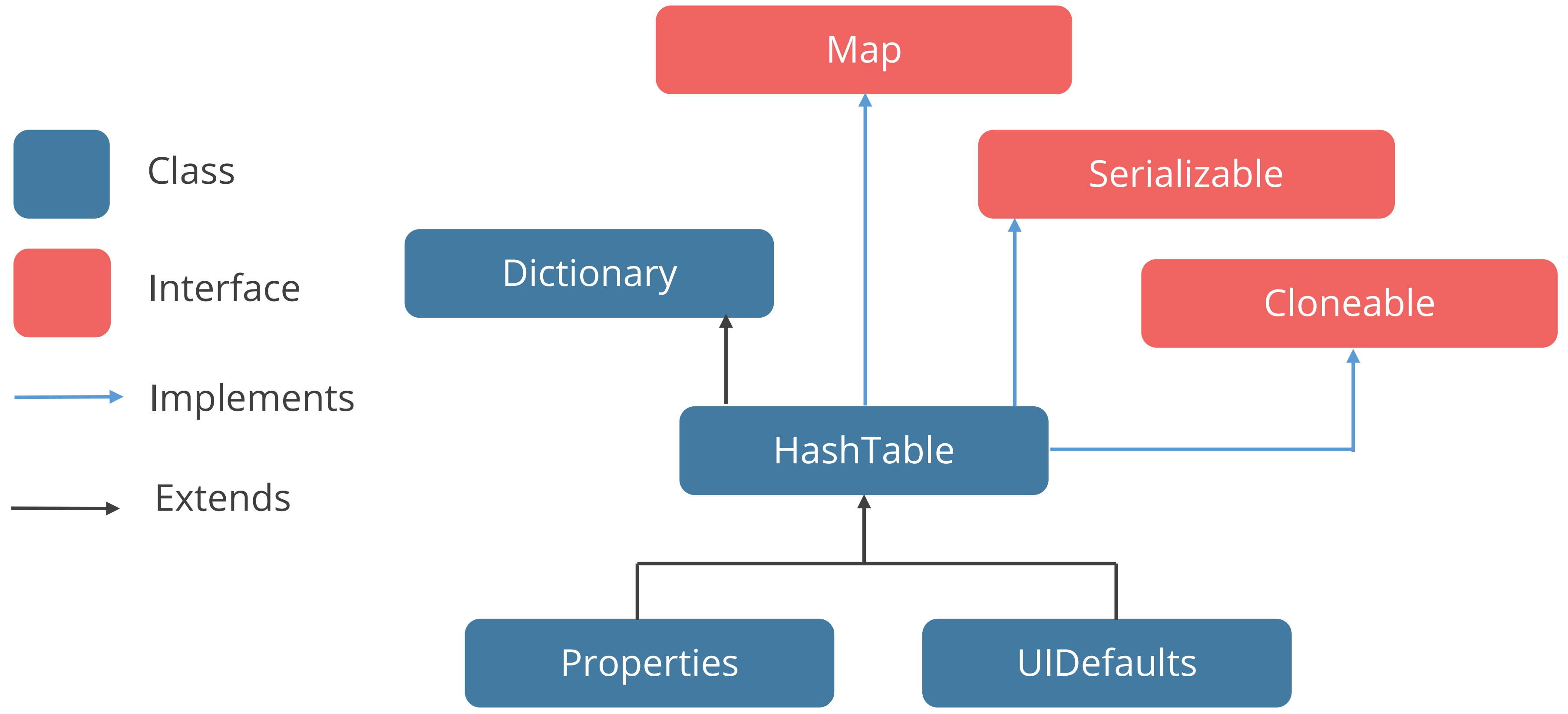
HashTable

This is a class for a hashtable that maps values to keys. To efficiently retrieve and store objects, keys must implement the equals and hashcode methods.

If a set does not hold the insertion order:

- It is synchronized.
- It keeps key or value pairs in a hash table.
- It helps in specifying an item that is used as a key.
- It offers a default ability of 11, and a LoadFactor of 0.75.
- It gives no long fail-speedy enumeration.

HashTable



Syntax:

```
public class Hashtable<K,V> extends
Dictionary<K,V> implements Map<K,V>,
Cloneable, Serializable
```

Here, K is the key type, and V is the mapped value.

Implementing HashMap and Hashtable

Problem Statement:

You have been asked to demonstrate how to use HashMap, LinkedHashMap, and Hashtable in Java for efficient data handling.

Outcome:

By demonstrating the use of HashMap, LinkedHashMap, and Hashtable in Java, you will learn efficient key-value data handling. HashMap offers fast access with no order, LinkedHashMap maintains insertion order, and Hashtable ensures thread safety, helping you choose the right map for your needs.

Note: Refer to the demo document for detailed steps: 05_Implement_HashMap_and_Hashtable

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a new project
2. Use a HashMap
3. Execute the code with example data
4. Sort the data based on keys
5. Iterate the data structure with example data
6. Use the remove() method and execute the code
7. Implement iteration to obtain all the keys from the map
8. Execute the entrySet() method and iterate through the code

TECHNOLOGY

Performance and Complexity

Performance and Complexity

Performance refers to how efficiently a software system or component executes tasks, such as a data structure. It is typically measured in terms of speed (execution time) and resource usage (such as memory and CPU cycles).

Complexity, specifically computational complexity, relates to the analysis of algorithms and data structures concerning the amount of computational resources they require. Complexity is usually expressed in terms of time and space

Performance and Complexity

There are many software developers today who are only familiar with the basic and widely used data structures, such as:

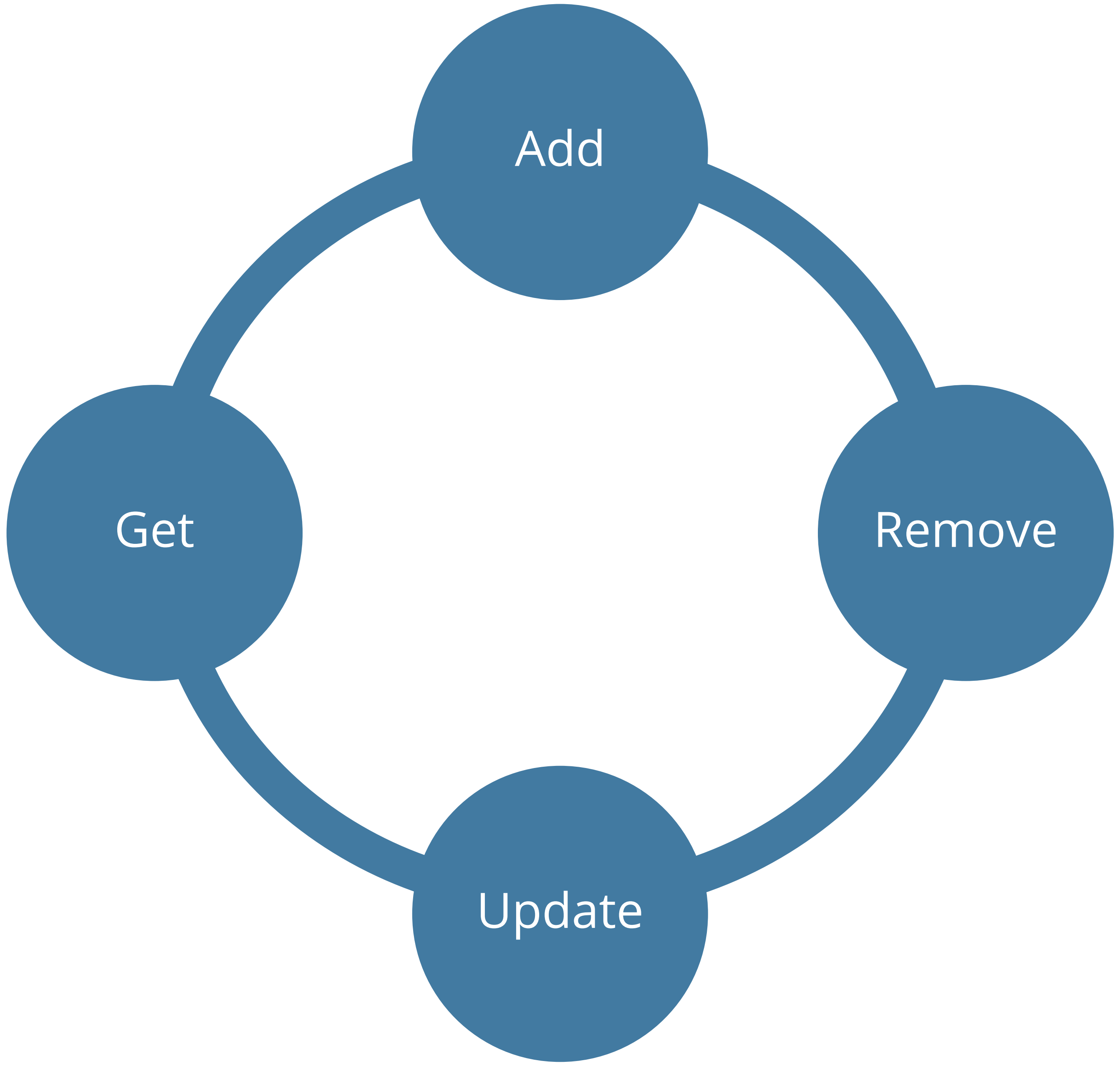
- Array
- Map
- Linked List

These structures are foundational and versatile, often sufficient for meeting the performance and complexity requirements of commercial software development.

Performance and Complexity

Big O notation

Describes how time or space will grow as the size of the input grows to do tasks like:



Performance and Complexity

Common Java collections complexities:

Number of stored elements in the table

	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Red-black tree
HashMap	-	-	O(1)	O(1)	O(h / n)	Hash Table

Here, **h** is the hashtable capacity or size.

Key Takeaways

- Type wrapper is training that encapsulates a primitive kind inside an object.
- The methods of the applet class required to display photos are public URL getDocumentBase() and public URL getCodeBase.
- When opting for Swing in AWT, we can use a JApplet containing the Swing controls.
- The mouseDragged() technique of MouseMotionListener is used to perform painting operations in an applet.
- Java.applet.AppletContext class facilitates communication between applets.

Key Takeaways

- The Vector class implements a scalable array of objects. It falls under legacy lessons but is compatible with collections.
- TreeSet is a crucial implementation of the SortedSet interface in Java that makes use of a tree for storage.
- Hashmap is part of Java’s collection and is located in the Java.util package deal. It provides a simple implementation of the Map interface in Java.
- Big O notation is a way to estimate how much time or space will increase as the input size grows while performing operations such as adding, removing, or updating data structures.

TECHNOLOGY

Thank You