SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

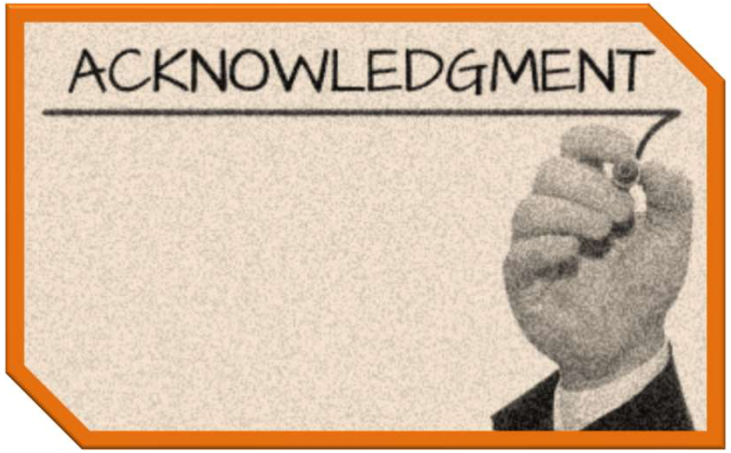
Unit – III

COLLECTIONS

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Need for Collection Framework
- Advantages of Collection Framework
- The Collection Interface
- The List Interface
- Collection Classes



- https://www.javatpoint.com/collections-in-java
- https://www.javadevjournal.com/java/java-arraylist/

Need for Collection Framework

- Before Collection Framework (or **before JDK 1.2**) was introduced
- The standard methods for grouping Java objects (or collections) were **array or Vector or Hashtable.**
- All three of these collections had **no common interface.**
- So, it was very **difficult for programmers to write algorithm** that can work for all kind of collections.
- Another drawback is that, most **of the 'Vector' methods are final.**
- So, we cannot **extend 'Vector' class** to implement a similar kind of collection.

Introduction

- A **Collection** is a group of **individual objects represented** as a **single unit.**
- The Collection interface **(java.util.Collection)** and Map interface **(java.util.Map)** are two main root interfaces of **Java collection classes.**
- The **Java collections** framework **(JCF)** is a set of classes and interfaces that implement commonly reusable **collection** data structures.
- Although referred to as a **framework**, it works in a manner **of a library.**
- The JCF provides both **interfaces** that define various **collections** and **classes** that implement them.

Advantages of Collection Framework

- **Consistent API :** The API has basic set of interfaces like **Collection, Set, List, or Map.**
- All those classes (such as **ArrayList, LinkedList, Vector** etc) which implements, these interfaces have **some common set of methods.**

- **Reduces programming effort:** The programmer need not to worry about **design of Collection** rather than he can focus on **its best use in** his program.

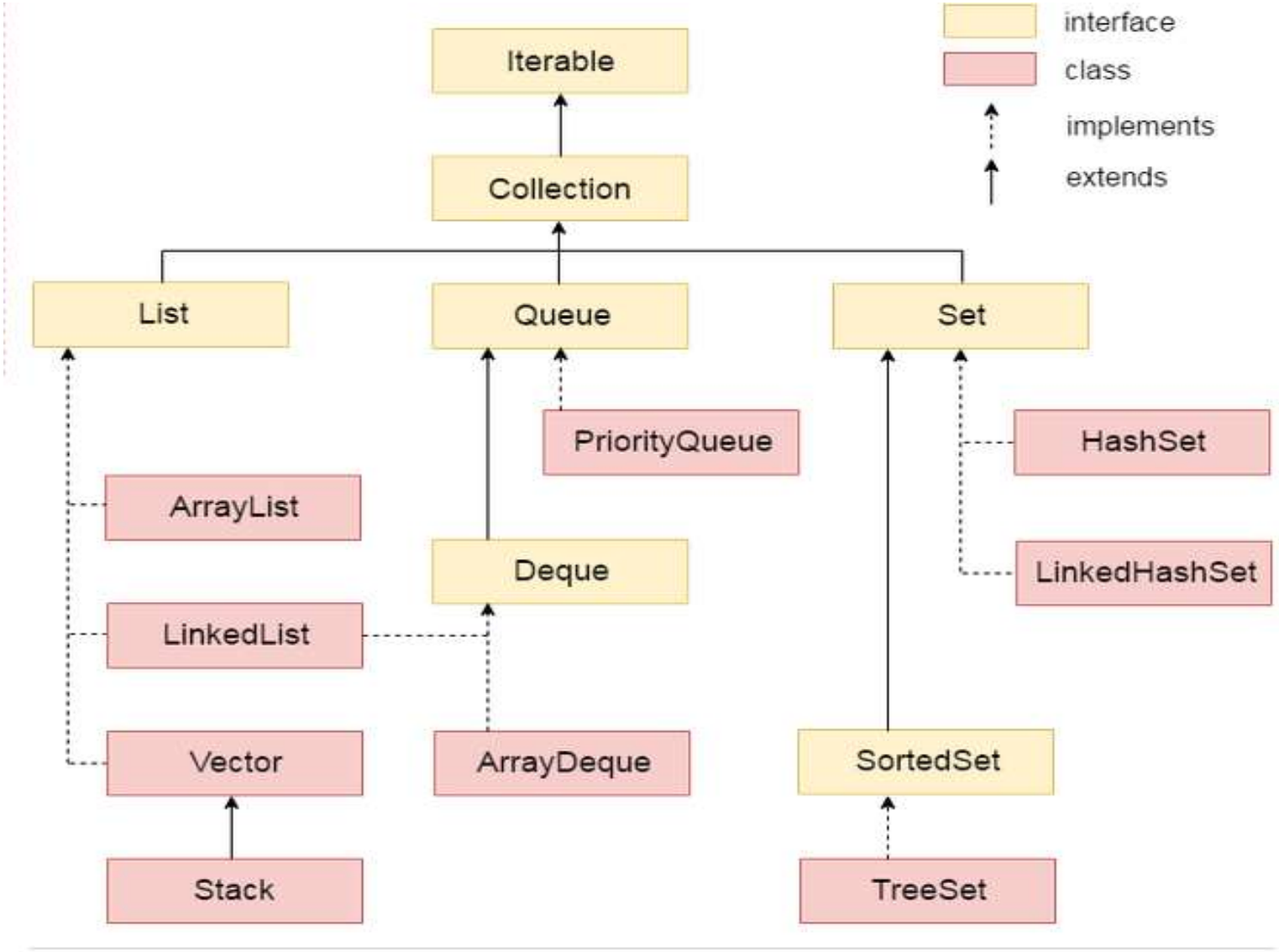
- **Increases program speed and quality:** Increases performance by providing high-performance implementations of useful **data structures and algorithms.**

JDK 5 Changed the Collections Framework

- When **JDK 5** was released,
- some **fundamental changes** were made to the **Collections Framework**

- These changes include the addition of
  - **Generics -** type safety.
  - **Autoboxing Facilitates - the Use of Primitive Types**
  - **The For-Each Style for Loop**

A Hierarchy of Collection Framework



The Collection Interface

- The **Collection interface is the foundation upon which the Collections Framework is built** because it must be implemented by any class that defines a collection.

- **Collection is a generic** interface that has this declaration:

  - **interface Collection<E>**

- Here, **E specifies the type of objects that the collection will hold.**

- **Collection extends the Iterable interface.**
- **This means that all collections can be cycled through by use of the foreach style for loop.**

The Methods Declared by Collection

➢**boolean add(E obj)**

- ✓ Adds obj to the invoking collection.
- ✓ Returns **true if** obj was added to the collection.
- ✓ Returns **false if obj is** already a member of the collection and
- ✓ the collection does not allow duplicates.

➢**boolean addAll(Collection<? extends E> c)**

- ✓ Adds all the elements of c to the invoking collection.
- ✓ Returns **true if the collection changed (i.e., the** elements were added). Otherwise, returns **false.**

➢**void clear( )**

- ✓ Removes all elements from the invoking collection.

The Methods Declared by Collection

- **boolean contains(Object obj)**

  Returns **true if obj is an element of the invoking** collection.
  Otherwise, returns **false.**

- **boolean containsAll(Collection<?> c)**

  Returns **true if the invoking collection contains all** elements of - c.
  Otherwise, returns **false.**

- **boolean equals(Object obj)**

  Returns **true if the invoking collection and obj are** equal.
  Otherwise, returns **false.**

The Methods Declared by Collection

- **boolean isEmpty( )**
  - Returns **true if the invoking collection is empty.**
  - Otherwise, returns **false.**

- **boolean remove(Object obj)**
  - Removes one instance of obj from the invoking collection.
  - Returns **true if the element was removed.**
  - Otherwise, returns **false.**

- **boolean removeAll(Collection<?> c)**
  - Removes all elements of c from the invoking collection.
  - Returns **true if the collection changed (i.e., elements** were removed). Otherwise, returns **false.**

- **default boolean removeIf(Predicate<? super E> predicate)**
  - Removes from the invoking collection those elements that satisfy the condition specified by predicate. **(Added by JDK 8.)**

The Methods Declared by Collection

- **boolean retainAll(Collection<?> c)**
  - Removes all elements from the invoking collection **except those in c.**
  - Returns **true if the collection changed** (i.e., elements were removed). Otherwise, returns **false.**

- **int size( )**
  - Returns the number of elements held in the invoking collection.

- **Object[ ] toArray( )**
  - Returns an array that contains all the elements stored in the invoking collection.
  - The array elements are copies of the collection elements.

Exception

- Several of these methods can throw an **UnsupportedOperationException**. this occurs if a collection cannot be modified.
- A **ClassCastException** is generated when one object is incompatible with another, such as when an attempt is made to add an incompatible object to a collection.
- A **NullPointerException** is thrown if an attempt is made to store a null object and null elements are not allowed in the collection.
- An **IllegalArgumentException** is thrown if an invalid argument is used.
- An **IllegalStateException** is thrown if an attempt is made to add an element to a fixed-length collection that is full.

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

The List Interface

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

The List Interface

- The List **interface extends Collection** and declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a **zero-based index**.
- A list may contain **duplicate elements**.
- List is a generic interface that has this declaration:
- interface List<E>
- Here, E specifies the type of objects that the list will hold.

The List Interface

- To obtain the object stored at a specific location, **call get( )** with the index of the object.
- To assign a value to an element in the list, **call set( ),** specifying the index of the object to be changed.
- To find the index of an object, use **indexOf( ).**
- You can obtain a sublist of a list by calling **subList( ),** specifying the beginning and ending indexes of the sublist
- List adds the methods **add(int, E)** and **addAll(int, Collection).**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Collection Classes

Array List

- Java ArrayList class **uses a dynamic array** **for storing** the elements.
- It inherits **Abstract List class and implements List interface.**

- The important points about Java ArrayList class are:
  - Java ArrayList class can contain **duplicate elements.**
  - Java ArrayList class maintains **insertion order.**
  - Java ArrayList allows **random access** because array works at the index basis.
  - In Java ArrayList class,
    - **Manipulation is slow**
    - because a lot of shifting needs to be occurred if any element is removed from the array list.

Arrays vs ArrayList

1. Array
   - has a fixed width and you need to specify the size upfront before you use it and its size will remain same even if you remove all elements from the Array.
   - On the other hand ArrayList is dynamic in nature and it will grow or shrink automatically as we add/ remove element from it.

2. Array is just data storage and you need to write the code to work on the Array (e.g. Add, remove or search etc.) While the ArrayList provides a set of methods which make it easy to work with it.

Constructors of Java ArrayList

- **ArrayList()**
  - It is used to build an empty array list.
- **ArrayList(Collection c)**
  - It is used to build an array list that is initialized with the elements of the collection c.
- **ArrayList(int capacity)**
  - It is used to build an array list that has the specified initial capacity.

Java Non-generic Vs Generic Collection

- Java collection framework was non-generic before JDK 1.5.
- Since 1.5, it is generic.
- Java new generic collection allows you to have only one type of object in collection.
- old non-generic example of creating java collection.
- ArrayList al=new ArrayList();//

Java Non-generic Vs Generic Collection

- New generic example of creating java collection.
- **ArrayList<String> al=new ArrayList<String>();**
- In generic collection, we specify the **type** in angular braces.
- Now ArrayList is forced to have only **specified type** of objects in it.
- If you try to add another type of object, **it gives compile time error**

Example 1	Example 2

To array	User DefinedList

- We have discussed :
  - Need for Collection Framework
  - Advantages of Collection Framework
  - The Collection Interface
  - The List Interface
  - Collection Classes
    - Array List

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- ✓ Accessing a Collection via an Iterator
- ✓ Methods Declared by Iterator
- ✓ The For-Each Version of the for Loop

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

COLLECTIONS - II

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Topics for today's discussion

- Accessing a Collection via an Iterator
- Methods Declared by Iterator
- Methods Declared by ListIterator
- The For-Each Version of the for Loop
- The Set Interface
- The SortedSet Interface
- Hash Set class
- Tree Set class

Accessing a Collection via an Iterator

- Creating a collection - want to cycle through the elements in a collection.
- Example
  - you might want to display each element.

- One way to do this is to employ an **iterator**,

- **Iterator** - which is an object that implements either the **Iterator or** the **ListIterator interface.**

- **Iterator enables you to cycle through**
  - **a collection,**
  - obtaining or removing elements

- **ListIterator extends Iterator** to allow bidirectional traversal of a list, and the modification of elements.

Accessing a Collection via an Iterator

- **Iterator and ListIterator are generic interfaces which are declared as shown here:**
  - interface Iterator<E>
  - interface ListIterator<E>

- **E specifies the type of objects being iterated.**

Methods Declared by Iterator

Method	Description
default void forEachRemaining(Consumer<? super E> *action*)	The action specified by *action* is executed on each unprocessed element in the collection.
boolean hasNext( )	Returns **true** if there are more elements. Otherwise, returns **false**.
E next( )	Returns the next element. Throws **NoSuchElementException** if there is not a next element.
default void remove( )	Removes the current element. Throws **IllegalStateException** if an attempt is made to call **remove( )** that is not preceded by a call to **next( )**. The default version throws an **UnsupportedOperationException**.

Methods Declared by ListIterator

boolean hasNext( )	Returns **true** if there is a next element. Otherwise, returns **false**.
boolean hasPrevious( )	Returns **true** if there is a previous element. Otherwise, returns **false**.
E next( )	Returns the next element. A **NoSuchElementException** is thrown if there is not a next element.
int nextIndex( )	Returns the index of the next element. If there is not a next element, returns the size of the list.
E previous( )	Returns the previous element. A **NoSuchElementException** is thrown if there is not a previous element.
int previousIndex( )	Returns the index of the previous element. If there is not a previous element, returns –1.
void remove( )	Removes the current element from the list. An **IllegalStateException** is thrown if **remove( )** is called before **next( )** or **previous( )** is invoked.
void set(E *obj*)	Assigns *obj* to the current element. This is the element last returned by a call to either **next( )** or **previous( )**.

Using an Iterator

- Before you can access a collection through an iterator, you must obtain one.
- Each of the collection classes provides an **iterator( ) method that returns an** iterator to the start of the collection.
- By using this iterator object,
  - you can access each element in the collection,
  - one element at a time.
- In general
  - to use an iterator to cycle through the contents of a collection, follow three steps

Using an Iterator

- Obtain an iterator to the start of the collection by calling the collection's **iterator( ) method.**
- Set up a loop that makes a call to **hasNext( ). Have the loop iterate as** long as **hasNext( ) returns true.**
- Within the loop, obtain each element by calling **next( ).**

Example Program

Error

```
E:\slides\Java - 2021\Unit - III>javac iterator.java
iterator.java:24: error: cannot find symbol
  while(itr.hasPrevious())
          ^
  symbol:   method hasPrevious()
  location: variable itr of type Iterator
1 error
```

Error

```
E:\slides\Java - 2021\Unit - III>javac iterator.java
iterator.java:6: error: unexpected type
  ArrayList<int> list=new ArrayList<int>();//Creating arraylist
           ^
  required: reference
  found:    int
iterator.java:6: error: unexpected type
  ArrayList<int> list=new ArrayList<int>();//Creating arraylist
                            ^
  required: reference
  found:    int
2 errors
```

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

The For-Each Version of the for Loop

The For-Each Version of the for Loop

- A for-each style loop is designed to cycle through a collection of objects, such as an array, in strictly sequential fashion, from start to finish.

- The for-each style of **for is also referred to as the enhanced for loop.**

- The general form of the for-each version of the **for is shown here:**

  for(type itr-var : collection) statement-block

  - **type specifies the type**
  - **itr-var specifies the name of an iteration variable** that will receive the elements from a collection, one at a time, from beginning to end.
  - The collection being cycled through is specified by collection.
  - There are various types of collections that can be used with the **for, but the only type used - array.**

The For-Each Version of the for Loop

```
// Use a for-each style for loop.
class ForEach
{
public static void main(String args[])
 {
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = 0;
// use for-each style for to display and sum the values
for(int x : nums)
 {
System.out.println("Value is: " + x);
sum += x;
}
System.out.println("Summation: " + sum);
}
}
```

The output from the program is shown here:

```
Value is: 1
Value is: 2
Value is: 3
Value is: 4
Value is: 5
Value is: 6
Value is: 7
Value is: 8
Value is: 9
Value is: 10
Summation: 55
```

The For-Each Version of the for Loop

- There is one important point to understand about the for-each style loop.
- Its iteration variable is **"read-only"** as it relates to the underlying array.
- **An assignment to the iteration variable has no effect** on the underlying array.
- In other words, you **can't change the contents of the array by assigning the iteration variable a new value**.

The For-Each Version of the for Loop

•For example, consider this program:

```
// The for-each loop is essentially read-only.
class NoChange
{
public static void main(String args[])
 {
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

for(int x: nums)
 {
System.out.print(x + " ");
x = x * 10; // no effect on nums
}

for(int x : nums)
System.out.print(x + " ");
}
}
```

The output, shown here, proves this point:

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

The Set Interface

The Set Interface

- The Set interface **defines a set.**
- It **extends Collection** and specifies the behavior of a collection that does **not allow duplicate elements.**
- Therefore, the add( ) method **returns false** if an attempt is made to add duplicate elements to a set.
- It does **not specify any additional methods** of its own.

- Set is a generic interface that has this declaration:

  – interface Set<E>

- Here, E specifies the type of objects that the set will hold.

The SortedSet Interface

- The **SortedSet interface extends Set** and declares the behavior of a set sorted in **ascending order.**
- SortedSet is a generic interface that has this declaration:

  » interface SortedSet<E>

- Here, E specifies the type of objects **that the set will** hold.
- In addition to those methods provided by Set, the SortedSet interface declares additional methods.
- Several methods throw a **NoSuchElementException** when no items are contained in the invoking set.

The SortedSet Interface

- SortedSet defines several methods that make set processing more convenient.

- To obtain the first object in the set, call **first( ).**
- To get the last element, use **last( ).**

- You can obtain a subset of a sorted set by calling **subSet(),** specifying the **first and last object** in the set.

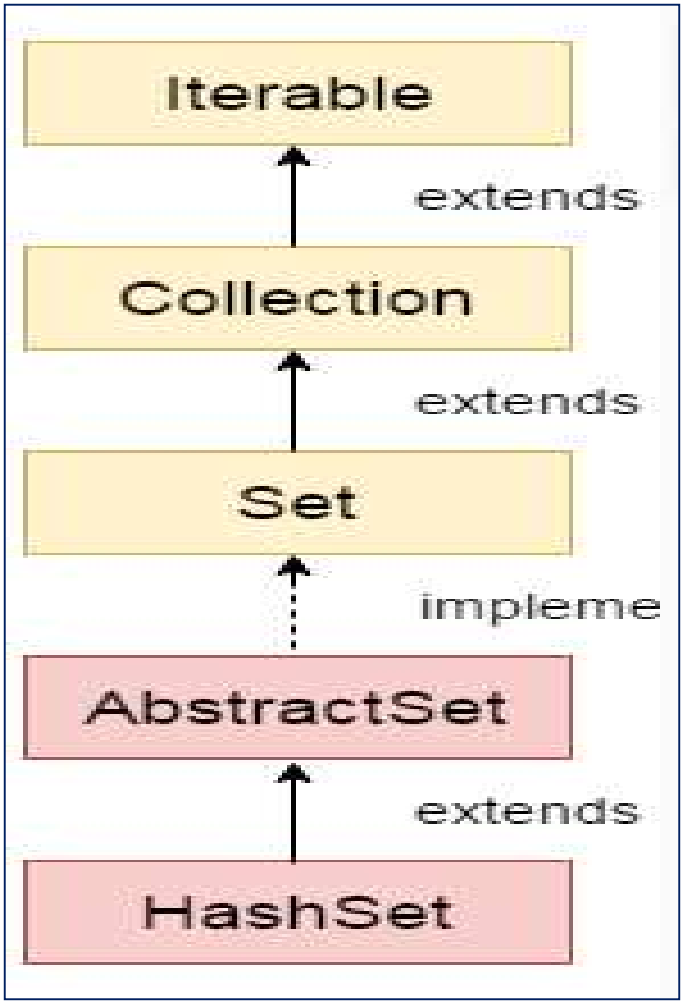
- If you need the subset that starts with the first element in the set, use **headSet( ).**
- If you want the subset that ends the set, use **tailSet( ).**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Hash Set class

HashSet class

- Java HashSet class is used to create a collection that uses a **hash table** for storage.
- It inherits the Abstract **Set class** and implements **Set interface.**
- The important points about **Java HashSet class are:**
- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains **unique elements only**.



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Java HashSet class

- Hash table stores information by using a mechanism called hashing.
- In hashing, the informational content of a key is used to determine a unique value, called its hash code.
- The hash code is then used as the index at which the data associated with the key is stored.
- The transformation of the key into its hash code is performed automatically—you never see the hash code itself.
- Also, your code can't directly index the hash table.
- The advantage of hashing is that it allows the execution time of **add(), contains(), remove(), and size() to remain constant even for large sets.**

Constructors of HashSet class

- **HashSet()**
  - It is used to construct a default HashSet.
- **HashSet(Collection c)**
  - It is used to initialize the hash set by using the elements of the collection c.
- **HashSet(int capacity)**
  - It is used to initialize the capacity of the hash set to the given integer value capacity.
  - The capacity grows automatically as elements are added to the HashSet.

Constructors of HashSet class

- **HashSet(int capacity,float fillratio)**
  - Load capacity
  - How full the hashset can be before it is resized

Example

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Tree Set class

TreeSet class

- Java TreeSet class implements the **Set interface** that uses a tree for storage.
- It inherits Abstract **Set class** and implements **NavigableSet interface.**
- The objects of TreeSet class are **stored in ascending order**.
- The important points about Java TreeSet class are:
  - Contains **unique elements only like HashSet.**
  - Access and retrieval times are quiet fast.
  - Maintains **ascending order.**

Constructors of TreeSet class

- **TreeSet()**
  - It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.

- **TreeSet(Collection c)**
  - It is used to build a new tree set that contains the elements of the collection c.

- **TreeSet(Comparator comp)**
  - It is used to construct an empty tree set that will be sorted according to given comparator.

Constructors of TreeSet class

- **TreeSet(SortedSet ss)**
  - It is used to build a TreeSet that contains the elements of the given SortedSet.

Example

We have discussed

- ✓ Accessing a Collection via an Iterator
- ✓ The For-Each Version of the for Loop
- ✓ The Set Interface
- ✓ The SortedSet Interface
- ✓ Hash Set class
- ✓ Tree Set class

- **Maps**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

COLLECTIONS - IV

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Maps
- Java HashMap class
- Java TreeMap class
- Comparator

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Maps

Map Interface

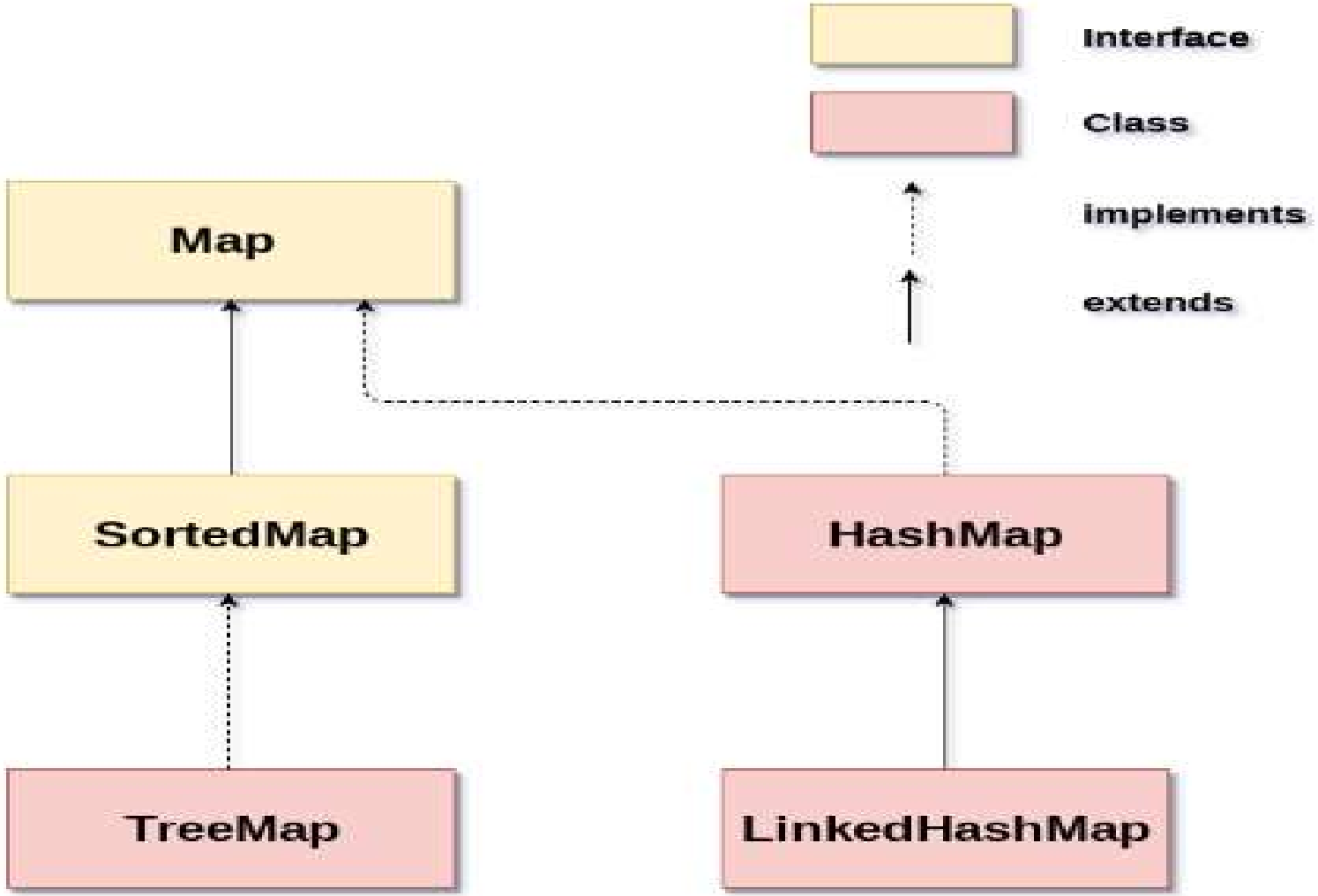
- A map contains values on the basis of key, **i.e. key and value pair.**
- Each key and value pair is known as **an entry.**
- A Map contains **unique keys.**
- A Map is useful
  - if you have to **search, update or delete** elements on the basis of a key.

Map Hierarchy

- For implementing Map in java:
- Two interfaces:
  - Map
  - SortedMap,

- Three classes:
  - HashMap
  - LinkedHashMap
  - TreeMap.

Java Map Hierarchy



Key Points

- A Map **doesn't allow duplicate keys**, but you can have duplicate values.
- HashMap and LinkedHashMap allow **null keys and values**
- TreeMap doesn't allow any **null key or value.**
- A Map **can't be traversed**
  - so you need to convert it into Set using keySet() or entrySet() method.

Class Description

- HashMap
  - HashMap is the implementation of Map
  - but it doesn't maintain any order

- TreeMap
  - TreeMap is the implementation of Map and SortedMap.
  - It maintains **ascending order.**

Useful methods of Map interface

Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
Set keySet()	It returns the Set view containing all the keys.
Set<Map.Entry<K,V>> entrySet()	It returns the Set view containing all the keys and values.
void clear()	It is used to reset the map.

Useful methods of Map interface

boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.

Useful methods of Map interface

V get(Object key)	This method returns the object that contains the value associated with the key.

boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.

int size()	This method returns the number of entries in the map.

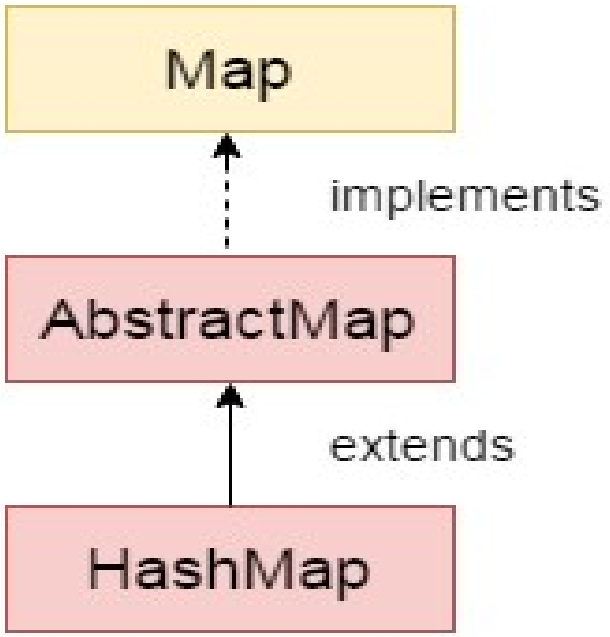
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Java HashMap class

Java HashMap class

- Java HashMap class implements the map interface by using a **hash table.**

- It inherits AbstractMap class and implements Map interface.



Key Points

- Java HashMap class contains values based on the key.
- Java HashMap class contains only unique keys.
- Java HashMap class may have one null key and multiple null values.

- Java HashMap class is non synchronized.
- Java HashMap class maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Constructors of HashMap class

- HashMap( )
  - a default hash map
- HashMap(Map<? extends K, ? extends V> m)
  - initializes the hash map by using the elements of m.
- HashMap(int capacity)
  - initializes the capacity of the hash map to capacity
- HashMap(int capacity, float fillRatio)
  - The default capacity is 16.
  - The default fill ratio is 0.75.

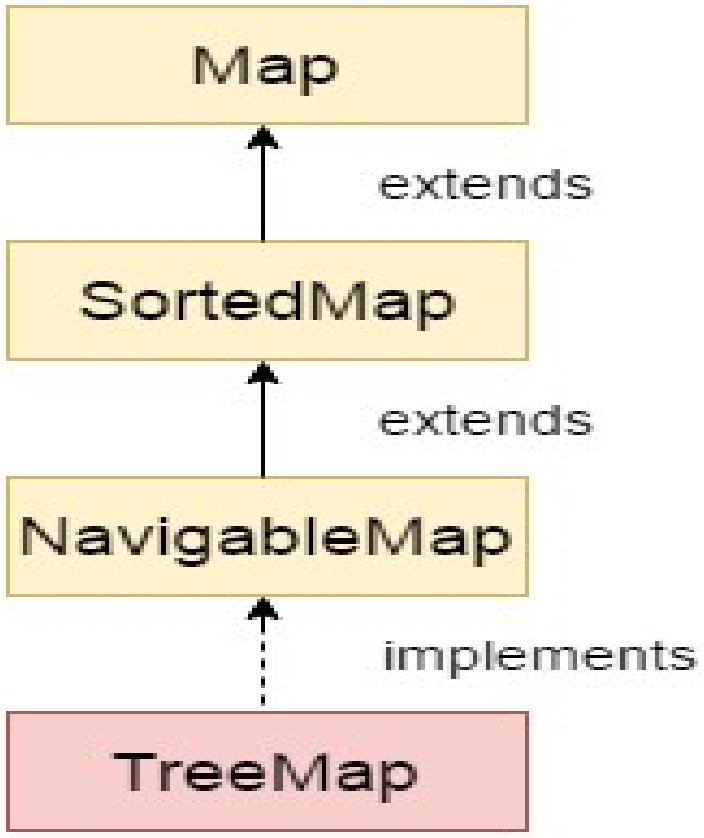
Example

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Java TreeMap class

Java TreeMap class

- Java TreeMap class provides an efficient means of storing key-value pairs in sorted order.



Constructors of TreeMap class

- TreeMap( )
  - constructs an empty tree map that will be sorted by using the natural order of its keys
- TreeMap(Comparator<? super K> comp)
  - constructs an empty tree-based map that will be sorted by using the Comparator comp.
- TreeMap(Map<? extends K, ? extends V> m)
  - initializes a tree map with the entries from m
- TreeMap(SortedMap<K, ? extends V> sm)
  - initializes a tree map with the entries from sm

Example

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Java Comparator interface

Java Comparator interface

- **Java Comparator interface** is used to order the objects of a user-defined class.

- This interface contains 2 methods
  - compare(Object obj1,Object obj2)
  - equals(Object element).

- It provides multiple sorting sequences,
  - you can sort the elements on the basis of any data member,
  - Employee - Empno, name, age or anything else.

Java Comparator interface

- The **compare( ) method compares two Objects**

  int compare(T obj1, T obj2)

- obj1 and obj2 are the objects to be compared.

- Normally, this method returns
  - **zero** if the objects are equal.
  - a **positive** value if obj1 is greater than obj2.
  - Otherwise, a **negative** value is returned.

Java Comparator interface

- The method can throw a **ClassCastException** **if the types of the objects are not compatible for** comparison.
- By implementing **compare(), you can alter the way that objects** are ordered.
- For example, **to sort in reverse order**, you can create a comparator that reverses the outcome of a comparison.

Java Comparator interface

- The **equals() method to test whether an object equals the** invoking comparator

  boolean equals(object obj)

- Here, obj is the object to be tested for equality.

- The method returns
  - **true if obj** and the invoking object are both **Comparator objects and use the same** ordering.
  - Otherwise, it returns **false.**

sorting List elements

- **public void sort(List list, Comparator c)**
  - is used to sort the elements of List by the given Comparator.

Example Program

We have discussed

- Java HashMap class
- Java TreeMap class
- Comparator

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Event Handling

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Event Handling - I

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- GUI / User Interface
- GUI elements
- Introducing the AWT
- Working with Frame Windows

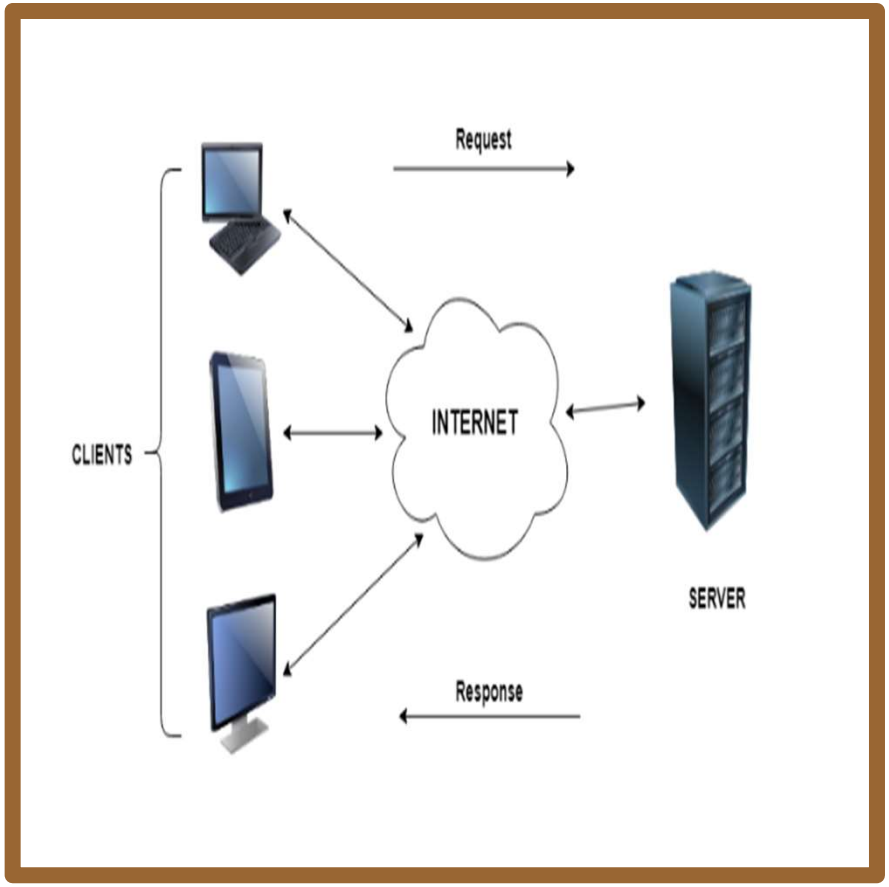
References / Acknowledgement

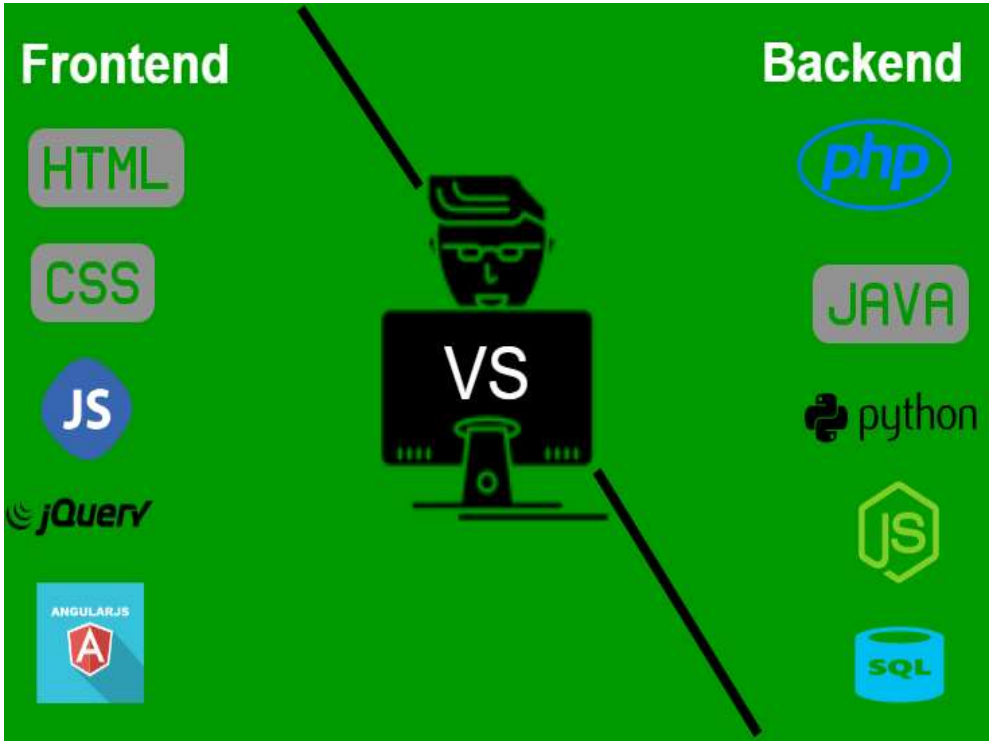
- https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

Review Question

- What is
  - Client – Server Computing
  - Front end
  - Back end

Client – Server Computing





https://www.tutorialspoint.com/Client-Server-Computing

https://www.geeksforgeeks.org/frontend-vs-backend/

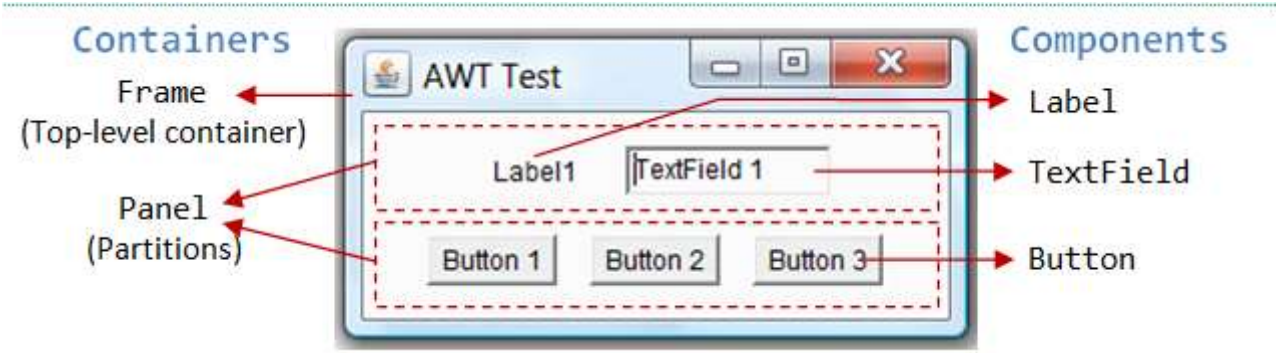
GUI / User Interface

- At the most basic level,
  - the user interface (UI) is the series of screens, pages, and visual elements—like buttons and icons—that enable a person to interact with a product or service

- users can view and interact with that data.

Login Screen – GUI - Applet

GUI elements



There are two types of GUI elements:

1. Component:
   - Components are elementary GUI entities, such as Button, Label, and TextField.
2. Container:
   - Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout).
   - A container can also hold sub-containers.

Java APIs for graphics programming:

- There are current three sets
  - AWT API was introduced in JDK 1.0. - obsolete and should be replaced by newer Swing components.
  - Swing API - was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1.
  - The latest JavaFX, which was integrated into JDK 8, is meant to replace Swing.

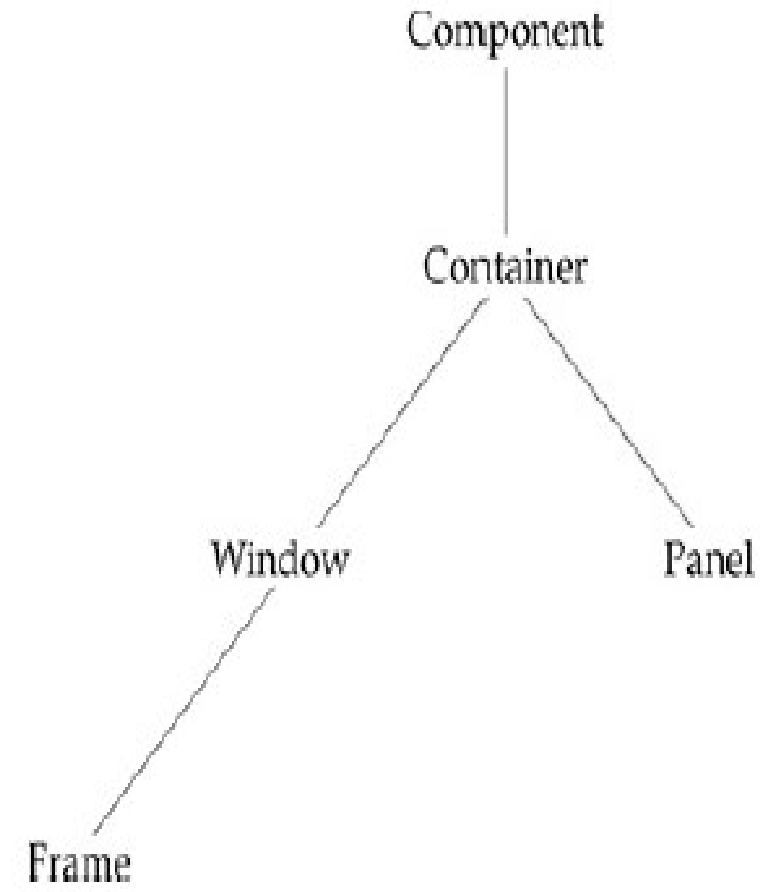

Frame Demo
SASTRA University

Introducing the AWT

- The Abstract Window Toolkit (AWT) was Java's first GUI framework, and it has been part of Java since version 1.0.
- It contains numerous classes and methods that allow you to create windows and simple controls.
- The AWT classes are contained in the **java.awt package.**
- **It is one of Java's** largest packages

Window Fundamentals

- The two most important window-related classes are **Frame and Panel.**
- **Frame encapsulates a top-level** window and it is typically used to create what would be thought of as a standard application window.



Component

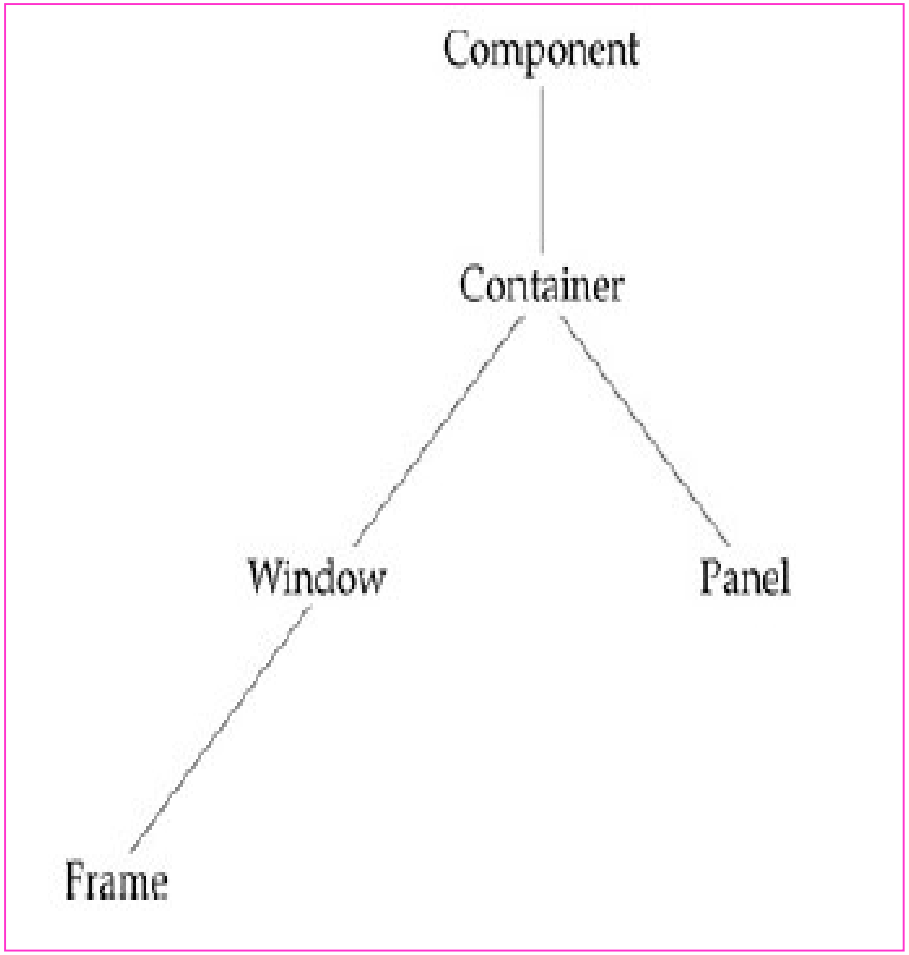
- At the top of the AWT hierarchy is the **Component class.**
- **Component is an** abstract class that encapsulates all of the attributes of a visual component.
- It defines over a hundred public methods that are responsible for managing events
  - such as mouse and keyboard input
  - positioning and sizing the window
  - and repainting.
- A Component object is responsible for remembering
  - the current foreground and background colors and
  - the currently selected text font.

Container

- The **Container class is a subclass of Component.**
- **It has additional methods** that allow other **Component objects to be nested within it.**
- A container is responsible for laying out (that is, positioning) any components that it contains.
- It does this through the use of various layout managers

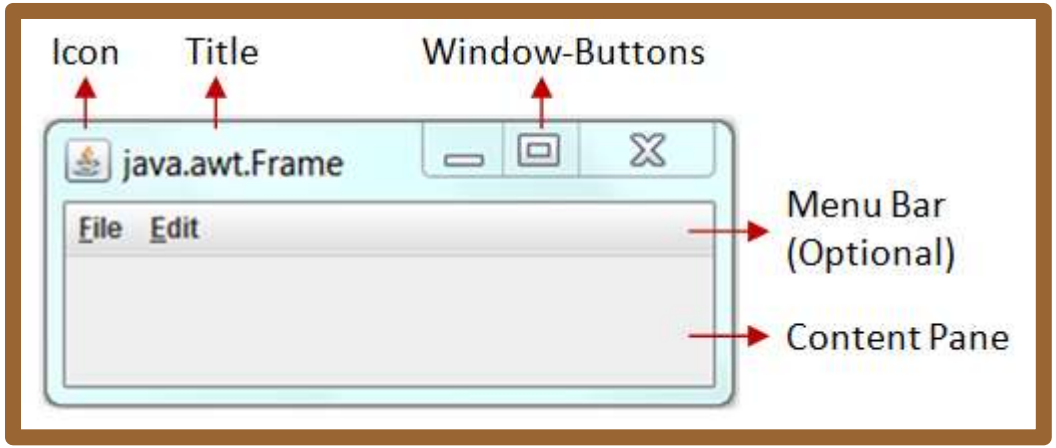
Window

- The **Window class creates a top-level window.**
- **A top-level window is not** contained within any other object; it sits directly on the desktop.
- Generally, you won't create **Window objects directly.**
- **Instead, you will use a subclass of Window called Frame**



Frame

- Frame encapsulates what is commonly thought of as a "window."
- **It is a** subclass of **Window**
  - **has a title bar**
  - **menu bar**
  - **borders and**
  - **resizing** corners.



- The precise look of a **Frame will differ among environments.**

Working with Frame Windows

- The type of AWT-based application window you will most often create is derived from **Frame.**

- **It creates a standard-style, toplevel** window that has all of the features normally associated with an application window, such as a close box and title.

- **Frame's constructors:**
  - Frame( ) throws HeadlessException
  - Frame(String title) throws HeadlessException

Working with Frame Windows

- You cannot specify the dimensions of the window.
- You must set the size of the window after it has been created.
- A **HeadlessException is thrown if an** attempt is made to create a **Frame instance in an environment that does not** support user interaction.

Several key methods

- void setSize(int *newWidth, int newHeight)*
- Dimension getSize( )
- void setVisible(boolean *visibleFlag)*
- void setTitle(String *newTitle)*

Example Program1

Other Methods

- To output a string to a **Frame, use drawString(), which is a member of the Graphics class.**
  - void drawString(String message, int x, int y)

- You can set both the foreground and background colors used by a **Frame – component**
  - void setBackground(Color *newColor)*
  - void setForeground(Color *newColor)*
  - Color getBackground( )
  - Color getForeground( )

Example Program2

How to close AWT Window in Java ?

- By anonymous class
- By inheriting WindowAdapter class
- By implementing WindowListener interface

Window Closing

https://www.javatpoint.com/java-close-awt-window

- GUI / User Interface
- GUI elements
- Introducing the AWT
- Working with Frame Windows

- Delegation event model
- Events
- Event sources
- Event classes
- Event Listeners


THANK
YOU

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Event Handling - II

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Delegation event model
- Events
- Event sources
- Event classes
- Event Listeners

References / Acknowledgement

- https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

Event handling – Demo Program

Adjustment Event

ItemEvent

Delegation event model

- The modern approach to handling events
  - is based on the delegation event model
  - which defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple:
  - a source generates an event and
  - sends it to one or more listeners.
- In this scheme
  - the listener simply waits until it receives an event.

Delegation event model

- Once received,
  - the listener processes the event and then returns.
- The advantage of this design
  - is that the application logic that processes events
  - is cleanly separated from the user interface logic that generates those events.
- A user interface element is able to "delegate" the processing of an event to a separate piece of code.

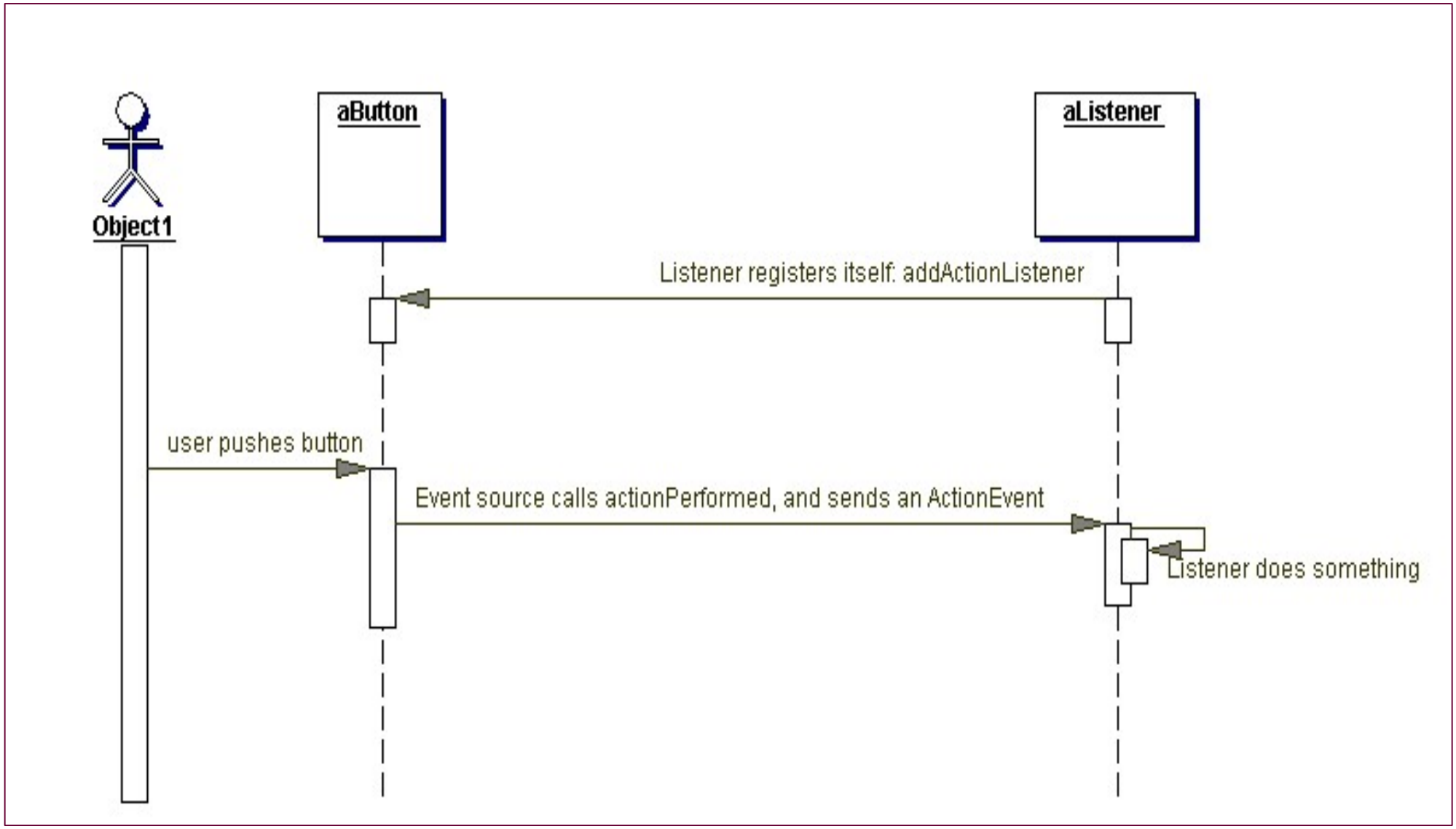
Delegation event model

- In the delegation event model,
  - listeners must register with a source in order to receive an event notification.
- This provides an important benefit:
  - notifications are sent only to listeners that want to receive them.
- Previously, an event was propagated up the containment hierarchy until it was handled by a component.
- This required components to receive events that they did not process, and it wasted valuable time.
- The delegation event model eliminates this overhead.

Event handling

- For the user to interact with a GUI
- The underlying operating system must support event handling.
  - Operating systems constantly monitor events such as keystrokes, mouse clicks, voice command, etc.
  - Operating systems sort out these events and report them to the appropriate application programs
  - Each application program then decides what to do in response to these events

An Example


Object1
aButton
aListener
Listener registers itself: addActionListener
user pushes button
Event source calls actionPerformed, and sends an ActionEvent
Listener does something

Events

- An event is an object that describes a state change in a source.
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Some of the activities that cause events to be generated are
  - pressing a button
  - entering a character via the keyboard
  - selecting an item in a list, and
  - clicking the mouse.

Events

- Events may also occur that are not directly caused by interactions with a user interface.

- For example,
  - an event may be generated when a timer expires
  - a counter exceeds a value
  - a software or hardware failure occurs, or
  - an operation is completed.

- Events can be defined as needed and appropriate by application.

Event sources

- A source is an object that generates an event.
- This occurs when the internal state of that object changes in some way.
- Sources may generate more than one type of event.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event.

Event sources

- Each type of event has its own registration method.
- General form is:
  - public void addTypeListener(TypeListener el)
- Here, Type is the name of the event and el is a reference to the event listener.
- For example,
  - The method that registers a keyboard event listener is called **addKeyListener()**.
  - The method that registers a mouse motion listener is called **addMouseMotionListener( )**.

Event Listeners

- When an event occurs, all registered listeners are notified and receive a copy of the event object.
- This is known as **multicasting** the event.
- In all cases, notifications are sent only to listeners that register to receive them.
- Some sources may allow only one listener to register.

- The general form is:
  - public void addTypeListener(TypeListener el) throws java.util.TooManyListenersException

- Here Type is the name of the event and el is a reference to the event listener.
- When such an event occurs, the registered listener is notified.
- This is known as **unicasting** the event.

Event Listeners

- A source must also provide a method that allows a listener to unregister an interest in a specific type of event.

- The general form is:
  - **public void removeTypeListener(TypeListener el)**
- Here, *Type* is the name of the event and *el* is a reference to the event listener.

- For example, to remove a keyboard listener, you would call **removeKeyListener()**.
- The methods that add or remove listeners are provided by the source that generates events.

- For example, the **Component** class provides methods to add and remove keyboard and mouse event listeners.

Event classes

- The Event classes that represent events are at the core of Java's event handling mechanism.

- Super class of the Java event class hierarchy is **EventObject**, which is in **java.util**. for all events.

- Constructor is :

  EventObject(Object *src*)

  - Here, *src* is the object that generates this event.

Event classes

- **EventObject** contains two methods:
  - **getSource( )** and
  - **toString( )**.

- The **getSource( )** method returns the source of the event.
  General form is : Object getSource( )

- The **toString( )** returns the string equivalent of the event.

Event classes

- EventObject is a superclass of all events.
- AWTEvent is a superclass of all AWT events that are handled by the delegation event model.
- The package **java.awt.event** defines several types of events that are generated by various user interface elements.

Event Classes in java.awt.event

- ActionEvent:
  - Generated when a button is pressed, a list item is double clicked, or a menu item is selected.
- AdjustmentEvent:
  - Generated when a scroll bar is manipulated.
- ComponentEvent:
  - Generated when a component is hidden, moved, resized, or becomes visible.
- ContainerEvent:
  - Generated when a component is added to or removed from a container.

Event Classes in java.awt.event

- FocusEvent:
  - Generated when a component gains or loses keyboard focus.
- ItemEvent:
  - Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
- KeyEvent:
  - Generated when input is received from the keyboard.

Event Classes in java.awt.event

- MouseEvent:
  - Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
- TextEvent:
  - Generated when the value of a text area or text field is changed.
- WindowEvent:
  - Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listeners

- A listener is an object that is notified when an event occurs.

- Event has two major requirements.
  - It must have been registered with one or more sources to receive notifications about specific types of events.
  - It must implement methods to receive and process these notifications.

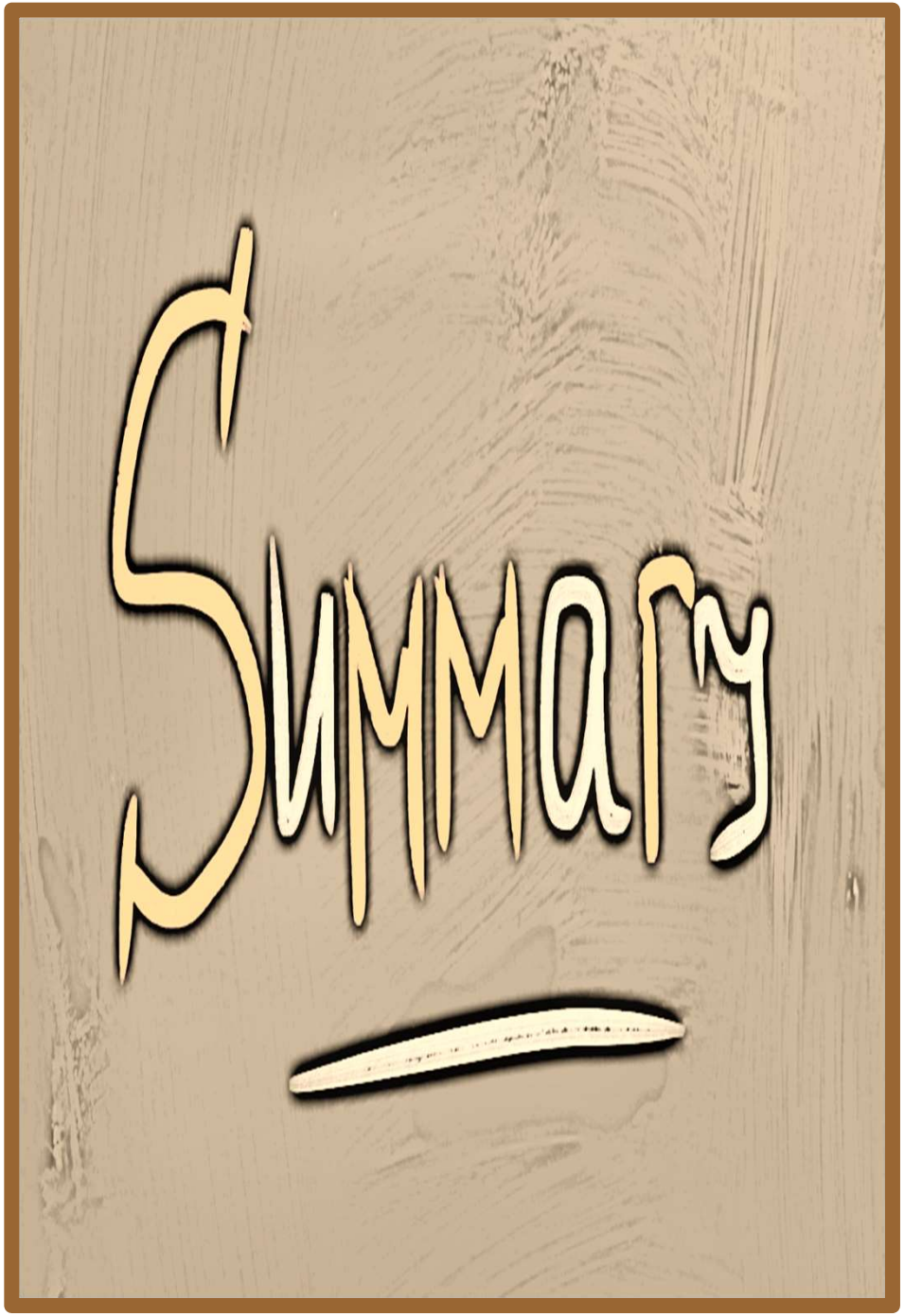
Event Listeners

- The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**.
- For example,
  - the **MouseMotionListener** interface defines two methods to receive notifications when the mouse is dragged or moved.
- Any object may receive and process one or both of these events if it provides an implementation of this interface.

Action Event Demo

Program

- Delegation event model
- Events
- Event sources
- Event classes
- Event Listeners



- Handling mouse events
- Handling keyboard events
- Adapter classes


THANK YOU!

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Event Handling - III

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Handling mouse events
- Handling keyboard events

Handling mouse events

- Mouse events can be handled by implementing
  - The **MouseListener** and
  - The **MouseMotionListener** interfaces.

Handling mouse events

- **MouseListener Interface** defines five methods.
- The general forms of these methods are:
  1. void mouseClicked(MouseEvent me)
  2. void mouseEntered(MouseEvent me)
  3. void mouseExited(MouseEvent me)
  4. void mousePressed(MouseEvent me)
  5. void mouseReleased(MouseEvent me)

Example

Handling mouse events

- **MouseMotionListener Interface** defines two methods.
- Their general forms are :
  1. void mouseDragged(MouseEvent me)
  2. void mouseMoved(MouseEvent me)

Example

Handling keyboard events

- Keyboard events, can be handled by implementing the **KeyListener** interface.
- **KeyListner** interface defines three methods.
- The general forms of these methods are :
  1. void keyPressed(KeyEvent ke)
  2. void keyReleased(KeyEvent ke)
  3. void keyTyped(KeyEvent ke)

Example

Multiple Events Demo

Sources of Events

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Classes

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

The ActionEvent Class

- **An ActionEvent is generated when**
  - **a button is pressed**
  - **a list item is doubleclicked**
  - **a menu item is selected**

- **String getActionCommand()**
  - obtain the command name for the invoking **ActionEvent object**

- **long getWhen()**
  - returns the time at which the event took place.
  - This iscalled the event's timestamp.

The ActionListener Interface

- This interface defines the **actionPerformed( ) method that is invoked when an** action event occurs.

- Its general form is shown here:
  - **void actionPerformed(ActionEvent *ae*)**

The AdjustmentEvent Class

- An **AdjustmentEvent is generated by a scroll bar**
- There are five types of adjustment events.
- The **AdjustmentEvent class defines integer constants that** can be used to identify them.

BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged.
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

The AdjustmentEvent Class

- **Adjustable getAdjustable()**
  - returns the object that generated the event
- **int getValue()**
  - when a scroll bar is manipulated, this method returns the value represented by that change.

The AdjustmentListener Interface

- This interface defines the **adjustmentValueChanged() method that is invoked** when an adjustment event occurs.

- Its general form is shown here:
  - **void adjustmentValueChanged(AdjustmentEvent *ae)***

The ItemEvent Class

- An **ItemEvent is generated when a**
  - **check box**
  - **list item is clicked**
  - checkable menu item
    - is selected or deselected.

- **Object getItem()**
  - can be used to obtain a reference to the item that changed

- **int getStateChange()**
  - method returns the state change

The ItemListener Interface

- This interface defines the **itemStateChanged( ) method that is invoked when** the state of an item changes.

- Its general form is shown here:
  - **void itemStateChanged(ItemEvent *ie)***

The TextEvent Class

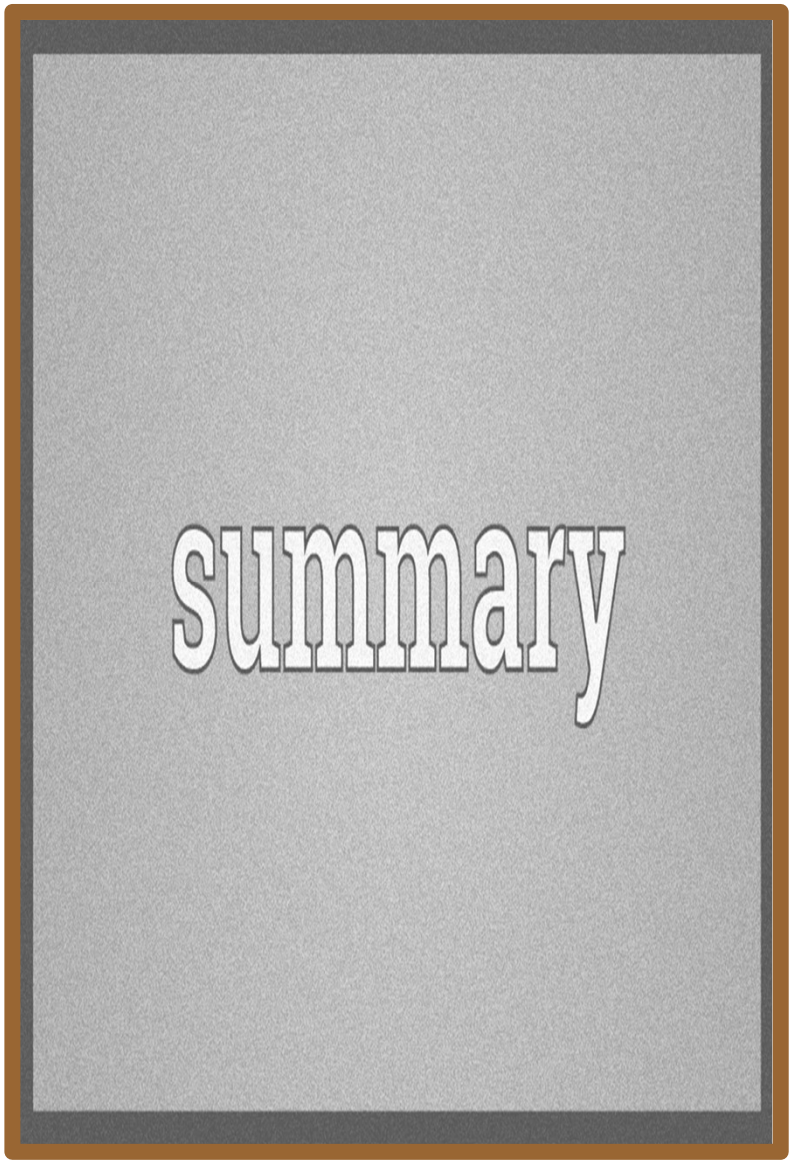
- Instances of this class describe text events.
- These are generated by text fields and text areas when characters are entered by a user or program.

The TextListener Interface

- This interface defines the **textValueChanged( ) method that is invoked when a** change occurs in a text area or text field.

- Its general form is shown here:
  - **void textValueChanged(TextEvent *te)***

- How to handle
  - mouse events
  - keyboard events



- Use of Adapter classes
- Anonymous Inner Class


THANK
YOU

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Event Handling - IV

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Adapter classes
- Anonymous Class

Some Key AWT GUI Concepts

- To demonstrate the fundamentals of event handling
  - use several simple, GUI-based programs

- Most events to which your program will respond will be generated by user interaction with GUI programs
- There are four key AWT features used - programs.
- First,
  - All create a top-level window by extending the Frame class.
  - Frame defines what one would think of as a “normal” window.
  - For example, it has minimize, maximize, and close boxes.

Some Key AWT GUI Concepts

- Second,
  - All override the paint( ) method to display output in the window.
  - This method is called by the run-time system to display output in the window.
  - For example,
    - it is called when a window is first shown and after a window has been hidden and then uncovered.

Some Key AWT GUI Concepts

- Third,
  - when your program needs output displayed
  - it does not call paint( ) directly.
  - Instead, you call repaint( ).
  - In essence, repaint( ) tells the AWT to call paint( ).
- Finally,
  - when the top-level Frame window for an application is closed
  - for example, by clicking its close box—the program must explicitly exit, often through a call to System.exit( ).
  - Clicking the close box, by itself, does not cause the program to terminate.
  - Therefore, it is necessary for an AWT-based GUI program to handle a window-close event.

Need for an adapter class?

Adapter classes

- Java provides a special feature, called an adapter class, that can simplify the creation of event handlers.
- An adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

Adapter classes

- Adapter classes in **java.awt.event are**.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener
?	Action Listener

Example

Anonymous InnerClass

- A class that have no name is known as anonymous inner class in java.

Example

Review Question - 1

Which of the following is NOT a method of the Thread class in Java?

A. isAlive()

B. getPriority()

C. getNames()

D. sleep()

Review Question - 2

- Which of the following method can be used to know the priority of a thread?

A. getPriority()

B. priority()

C. isRunning()

D. getThreadPriority()

Review Question - 3

- Which of the following is NOT a way to read data from the keyboard?

A. Using an object of DataInputStream Class

B. Using an object of Scanner Class

C. Passing the data as command line input

D. Using an object of FileInputStream Class

Review Question - 4

- All the classes related to multithreading are defined in which of the following packages?

A. java.awt.*

B. java.io.*

C. java.net.*

D. None of the Above

Review Question - 5

- The initial default capacity of HashSet is

A. 0

B. 16

C. 12

D. 10

Review Question - 6

- Java TreeSet class

A. implements the Set interface

B. contains unique elements only

C. doesn't allow null element.

D. maintains ascending order.

Review Question - 7

- The initial default load factor of Java HashMap class

A. 16

B. 0.75

C. 12

D. 0.25

Review Question - 8

- ¥ Is "length".length() allowed syntax? If so, what is the returned value?

A. No,Invalid

B. Yes, it is valid and returns 6.

C. Yes, it is valid and returns 5.

D. Yes, it is valid and returns 0.

Review Question - 9

- String city = "Thanjavur";
- What is returned by city.charAt (2)?

A. h

B. a

C. u

D. v

Review Question - 10

What is the output of the following program?

```
class First
{
public static void main(String args[])
{
String s1="SASTRA";
String s2=new String("SASTRA");
System.out.println(s1==s2);
}
}
```

Review Question - 11

- An AdjustmentEvent is generated by

A. a scroll bar

B. a button

C. a Menu bar

D. a Menu item

Review Question - 12

- Which of these methods can be used to obtain the coordinates of a mouse?

A. getPoint()

B. getCoordinates()

C. getMouseXY()

D. getMouseCoordinates()

Review Question - 13

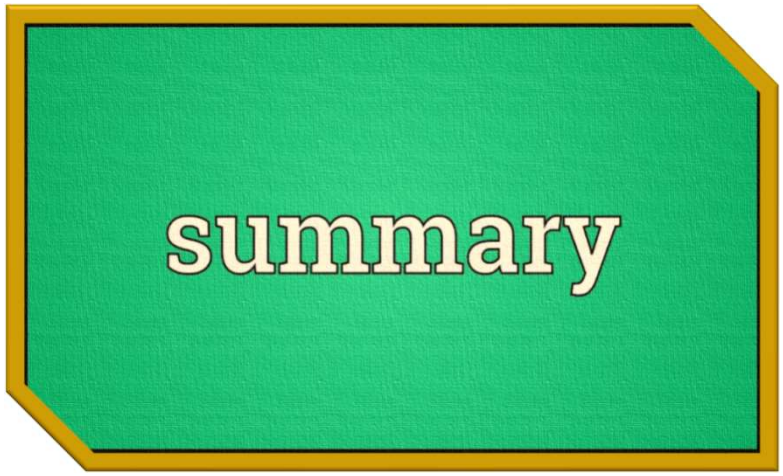
- Which of these interfaces define a method actionPerformed()?

A. ComponentListener

B. ContainerListener

C. ActionListener

D. InputListener



- Use of Adapter classes
- Anonymous Inner Class

- Unit IV
  - Swings


THANK
YOU