Collection

Session Objective

- Collection Introduction
- List
- Set
- Map
- Iterator
- Wrapper class
- Date & Time

Collection Introduction

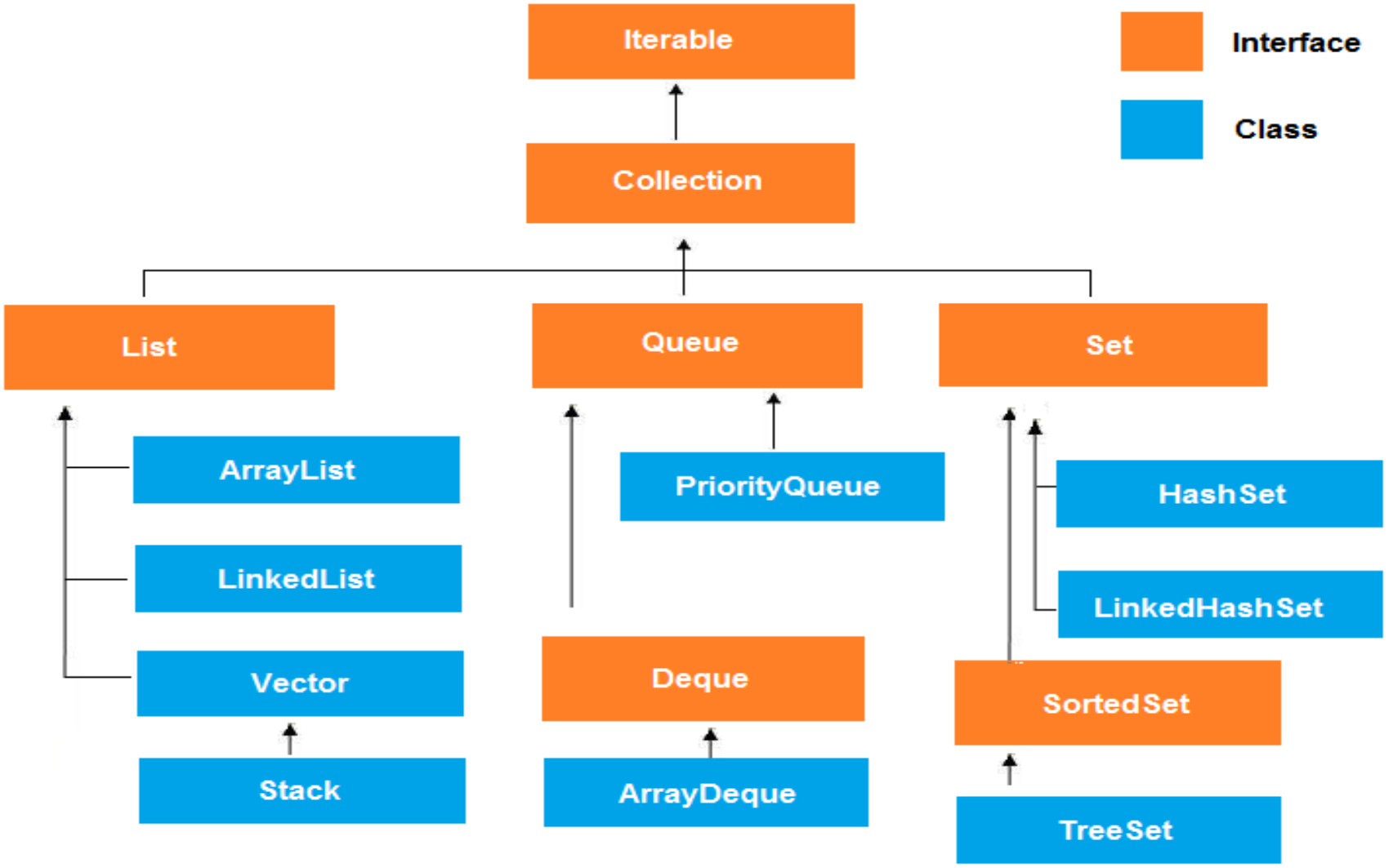
Collection Introduction

- A collection, is group of objects represented by one object.
- Java Collections framework consist of the interfaces and classes which helps in working with different types of collections such as lists, sets, maps, stacks and queues etc.

Need for Collection Framework :

- Before Collection Framework or before JDK 1.2 was introduced, the standard methods for grouping Java objects were Arrays, Vectors or Hashtables. All of these collections had no common interface.
- Accessing elements of these Data Structures was a hassle as each had a different method for accessing its members

Collection



Advantages of Collection Framework

- **Consistent API** : The API has a basic set of interfaces like Collection, Set, List, or Map. All classes like ArrayList, LinkedList, Vector, etc that implement these interfaces have some common set of methods.
- **Reduces programming effort**: A programmer doesn't have to worry about the design of Collection, and 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.

List

List

- It is an ordered collection of objects in which duplicate values can be stored.
- List Interface is implemented by the classes of ArrayList, LinkedList, Vector and Stack.
- Java.util.List is a child interface of Collection.

ArrayList Contd…

- An ArrayList in Java represent a resizable list of objects.
- Elements can be added, remove, find, sort and replace in this list.

ArrayList Contd…

ArrayList Feature

I. Ordered
II. Index based
III. Dynamic resizing
IV. Non synchronized
V. Duplicates allowed

ArrayList

Java 8 Collections:

Collections can be iterated in single line and perform an action on all elements of collection using forEach statement.

Example:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.forEach(System.out::print);
```


LinkedList Contd…

- Java LinkedList class is doubly-linked list implementation of the List and Deque interfaces.
- It implements all optional list operations, and permits all elements including null.

```
LinkedList<String> list=new LinkedList<String>();
    list.add("Steve");
    list.add("Carl");
    list.add("Raj");
    System.out.println(list);
```

LinkedList Contd…

LinkedList Features:

- LinkedList maintains the insertion order of the elements.
- It is not synchronized.
- ListIterator can be used to iterate LinkedList elements.

LinkedList

LinkedList Methods:

- ✓ boolean add(Object o)
- ✓ void add(int index, Object element)
- ✓ int size()
- ✓ boolean remove(Object o)
- ✓ int indexOf(Object o)
- ✓ Object[] toArray()
- ✓ Iterator iterator()


.object.modifiers[
ntext.object.modifiers[
ntext.object.modif
.context.obje
.contex
.ops.object.mode_set
.ops.object.modifier_ap
.context.scene

Set

Set Interface

- It is an unordered collection of objects in which duplicate values cannot be stored.
- Set is implemented by HashSet, LinkedHashSet or TreeSet

HashSet

- Java HashSet class implements the Set interface, backed by a hash table.
- It is un-ordered collection and makes no guarantees as to the iteration order of the set.
- Duplicate values are not allowed in HashSet.

```
HashSet<String> hset =  new HashSet<String>();
    hset.add("Apple");
    hset.add("Mango");
    hset.add("Grapes");
    System.out.println(hset);
```

HashSet

HashSet Features:

- One NULL element is allowed in HashSet.
- This class offers constant time performance for the basic operations like add, remove, contains and size.
- HashSet is not synchronized.

TreeSet

- Java TreeSet class extends AbstractSet .
- It is very similar to HashSet class, except it stores the element in sorted order.
- The sort order is either natural order or by a Comparator provided at treeset.

```
TreeSet<String> tset = new TreeSet<String>();
        tset.add("String");
        tset.add("Test");
        tset.add("Pen");
        System.out.println(tset);
```

TreeSet Features

- Duplicate values are not allowed in TreeSet.
- NULL is not allowed in TreeSet.
- It is an ordered collection which store the elements in sorted order.
- Like HashSet, this class offers constant time performance for the basic operations add, remove, contains and size.
- TreeSet is not synchronized.


.object.modifiers[
ntext.object.modifier
ntext.object.modif
.contex
.ops.object.mode_set
.ops.object.modifier_ap
.context.scene

Map

Map

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

HashMap Contd…

- Java HashMap class contains values based on the key.
- It contains only unique keys.
- It may have one null key and multiple null values.
- It is non synchronized.
- It maintains no order.
- The initial default capacity of Java HashMap class is 16

HashMap

Methods of HashMap

- Set entrySet()
- Set keySet()
- V put(Object key, Object value)
- V remove(Object key)
- void putAll(Map map)
- void clear()

Hashtable

- A Hashtable is an array of a list.
- It contains unique elements.
- It doesn't allow null key or value.
- It is synchronized.
- The initial default capacity of Hashtable class is 11

TreeMap

- Java TreeMap provides an efficient way of storing key-value pairs in sorted order.
- It contains only unique elements.
- It cannot have a null key but can have multiple null values.
- It is non synchronized.
- It maintains ascending order.

Map Demo

```
HashMap<Integer,String> map=new HashMap<Integer,String>();
      map.put(101,"Thomas");
      map.put(106,"Hendry");
      map.put(103,"Ben");
      map.put(120,"Sam");
      for(Entry r:map.entrySet()){
     System.out.println(r.getKey()+"    "+r.getValue());
      }
      Set s=map.keySet();
      System.out.println(s);
      Collection c=map.values();
      System.out.println(c);
```


Iterator

Iterator Contd…

- Iterator can be used to perform both read and remove operations.
- Iterator is used when all the element needs to be enumerated in Collection framework which implements interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface.

Iterator

// Returns true if the iteration has more elements

public boolean hasNext();

// Returns the next element in the iteration

// It throws NoSuchElementException if no more is element present

public Object next();

// Remove the next element in the iteration

// This method can be called only once per call to next()

public void remove();

ListIterator Contd…

- It is only applicable for List collection implemented classes like arraylist, linkedlist etc.
- It provides bi-directional iteration.
- ListIterator object can be created by calling listIterator() method present in List interface.

List Iterator

Backward direction

// Returns true if the iteration has more elements
// while traversing backward

public boolean hasPrevious();

// Returns the previous element in the iteration
// and can throws NoSuchElementException
// if no more element present

public Object previous();

Wrapper class

Wrapper Classes Contd…

A Wrapper class is a class whose object wraps or contains a primitive data types.

Need of Wrapper Classes

- They convert primitive data types into objects.
- Objects are needed to modify the arguments passed into a method because primitive types are passed by value.
- The classes in java.util package handles only objects.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects and not primitive types.

Wrapper Classes Contd…

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

Wrapper Classes Contd…

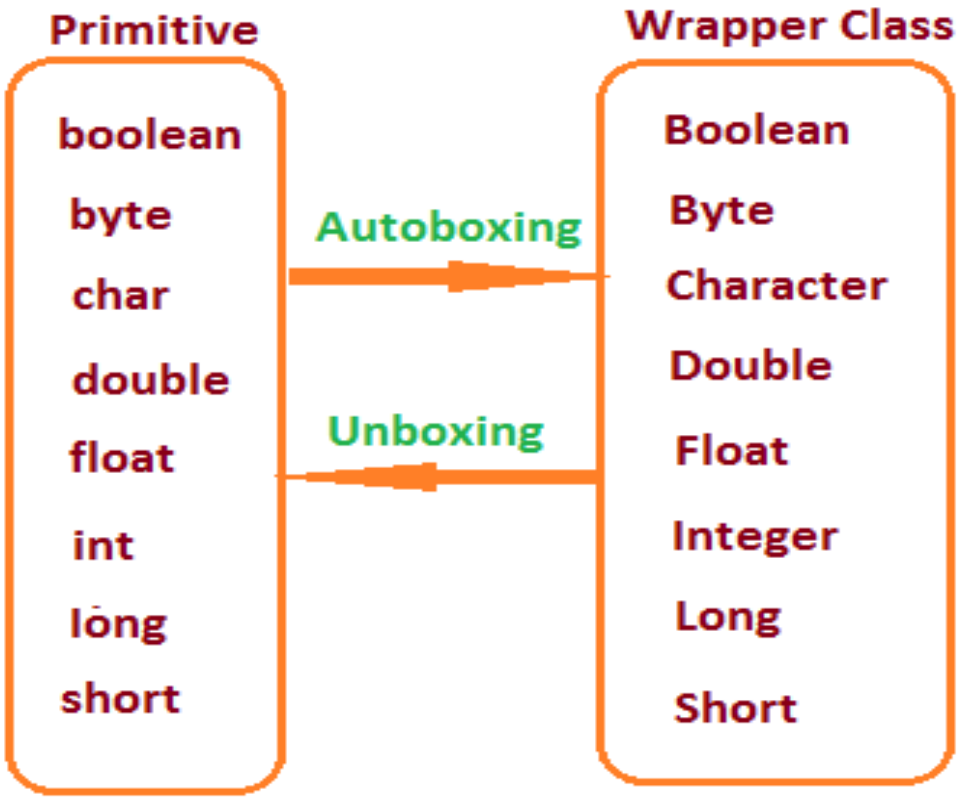
Autoboxing:

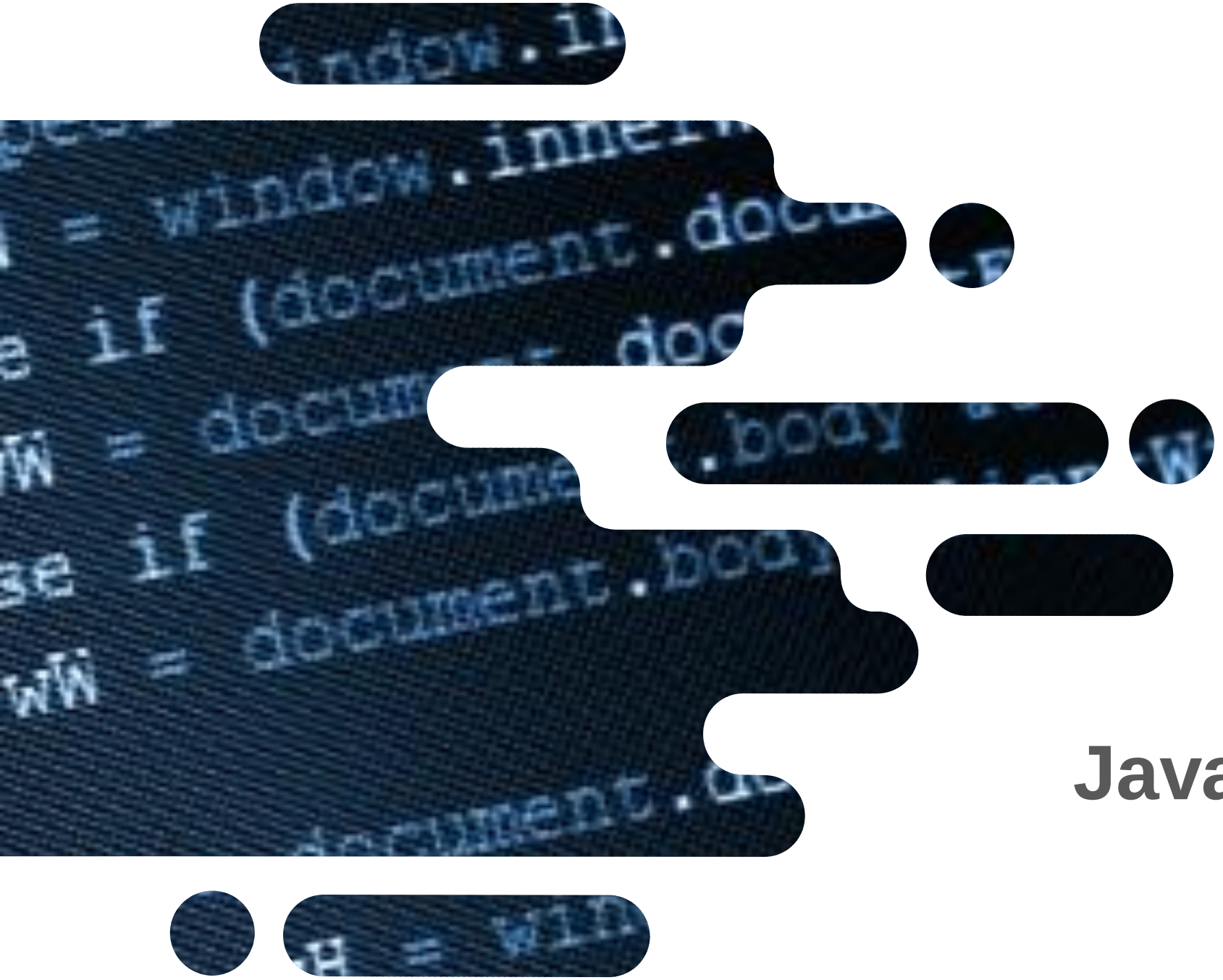
- Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.
- **For example** – conversion of int to Integer, long to Long, double to Double etc.

Unboxing:

- It is the reverse process of autoboxing.
- Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.
- **For example** – conversion of Integer to int, Long to long, Double to double etc.

Wrapper Classes Contd…





Java 8 Date & Time

Java 8 Date & Time Contd…

Issues with the Existing Date/Time APIs

- **Thread Safety** – The Date and Calendar classes are not thread
- **APIs Design and Ease of Understanding** – The Date and Calendar APIs are poorly designed with inadequate methods to perform day-to-day operations
- **ZonedDate and Time** – Developers had to write additional logic to handle timezone logic

Java 8 Date & Time Contd…

- The package **java.time** introduced a new date-time API, most important classes among them are :
- **Local:** Simplified date-time API with no complexity of timezone handling.
- **Zoned:** Specialized date-time API to deal with various timezones.

Java 8 Date & Time Contd…

LocalDate

- The LocalDate represents a date in ISO format (yyyy-MM-dd) without time.
- It can be used to store dates like birthdays and paydays.

LocalDate Contd…

- Current date can be created from the system clock as
  - LocalDate localDate = LocalDate.now();
- LocalDate representing a specific day, month and year can be obtained using the **“of”** method or by using the “parse” method
  - LocalDate.of(2015, 02, 20);
  - LocalDate.parse("2015-02-20");
- Current local date and adds one day
  - LocalDate tomorrow = LocalDate.now().plusDays(1);

LocalDate

- Relationship of a date
  - boolean notBefore = LocalDate.parse("2016-06-12")
  - .isBefore(LocalDate.parse("2016-06-11"));

  - boolean isAfter = LocalDate.parse("2016-06-12")
  - .isAfter(LocalDate.parse("2016-06-11"));

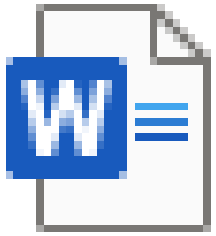
LocalTime

- Current LocalTime can be created from the system clock as
  - LocalTime now = LocalTime.now();
- Parsing a string representation
  - LocalTime localTime = LocalTime.parse("22:45:30", DateTimeFormatter.ofPattern("HH:mm:ss")); System.out.println(localTime);

LocalDateTime

- LocalDateTime can be obtained from the system clock
  - LocalDateTime.now();
  - LocalDateTime.of(2015, Month.FEBRUARY, 20, 06, 30);
  - LocalDateTime.parse("2015-02-20T06:30:00");

Collection – Practice questions



Collection -
Practice questions

thank you

www.hexaware.com