Dt : 15/11/2022

Note:

=>In realtime Stack<E> and Queue<E> are used in Algorithmic design part of Vendor engineering,which means Product developments.

==========

3.Queue<E>:

=>Queue<E> organizes elements based on the algorithm First-In-First-Out or Last-In-Last-Out.

=>The following are some important methods of Queue<E>:

public abstract boolean add(E);

public abstract boolean offer(E);

public abstract E remove();

public abstract E poll();

public abstract E element();

public abstract E peek();

=>"PriorityQueue<E>" is the implementation class of Queue<E> and which organizes elements based on elements-priority.

Ex-Program : DemoQueue.java

```
package maccess;
import java.util.*;
public class DemoQueue {
	@SuppressWarnings("removal")
	public static void main(String[] args) {
	    Queue<Integer> ob = new PriorityQueue<Integer>();
	    Scanner s = new Scanner(System.in);
	    try(s;){
	      try {
	           while(true) {
```

```
                    System.out.println("****Choice****");


System.out.println("1.add(E)\n2.remove()\n3.poll()\n4.element()\
n5.peek()\n6.exit");
                    System.out.println("Enter the Choice:");
                    switch(s.nextInt()) {
                    case 1:
                         System.out.println("Enter the ele:");
                         ob.add(new Integer(s.nextInt()));
                         System.out.println(ob.toString());
                         break;
                    case 2:
                         if(ob.isEmpty()) {
                              System.out.println("Queue is
empty...");
                         }else {
                              ob.remove();
                              System.out.println("Ele removed
from Queue...");
                              System.out.println(ob.toString());
                         }
                         break;
                    case 3:
                         if(ob.isEmpty()) {
                              System.out.println("Queue is
empty...");
                         }else {

                             ob.poll();
                              System.out.println(ob.toString());
                         }
                         break;
                    case 4:
                         if(ob.isEmpty()) {
                              System.out.println("Queue is
empty...");
                         }else {

     System.out.println("element:"+ob.element());
                              System.out.println(ob.toString());
                         }
                         break;
                    case 5:
                         if(ob.isEmpty()) {
                              System.out.println("Queue is
empty...");
```

```
                              }else {

      System.out.println("peek:"+ob.peek());
                                    System.out.println(ob.toString());
                              }
                              break;
                        case 6:
                              System.out.println("Operations stopped
on Queue...");
                              System.exit(0);
                        default:
                              System.out.println("Invalid Choice..");
                        }//end of switch
                  }//end of loop
            }catch(Exception e) {e.printStackTrace();}
         }//end of try
      }
}
```

o/p:

*****Choice*****

1.add(E)

2.remove()

3.poll()

4.element()

5.peek()

6.exit

Enter the Choice:

1

Enter the ele:

11

[11]

*****Choice*****

1.add(E)

2.remove()

3.poll()

4.element()

5.peek()

6.exit

Enter the Choice:

1

Enter the ele:

12

[11, 12]

*****Choice*****

1.add(E)

2.remove()

3.poll()

4.element()

5.peek()

6.exit

Enter the Choice:

1

Enter the ele:

13

[11, 12, 13]

*****Choice*****

1.add(E)

2.remove()

3.poll()

4.element()

5.peek()

6.exit

Enter the Choice:

5

peek:11

[11, 12, 13]

*****Choice*****

1.add(E)

2.remove()

3.poll()

4.element()

5.peek()

6.exit

Enter the Choice:

==========

faq:

define Deque<E>?

=>Deque<E> is an interface from from java.util package and which is extended from Queue<E>.

=>We perform operations on both ends of Deque<E> and which is also known as

Double-Ended-Queue.

=>The following are some important methods of Deque<E>:

public abstract void addFirst(E);

public abstract void addLast(E);

public abstract boolean offerFirst(E);

public abstract boolean offerLast(E);

public abstract E removeFirst();

public abstract E removeLast();

public abstract E pollFirst();

public abstract E pollLast();

public abstract E getFirst();

public abstract E getLast();

public abstract E peekFirst();

public abstract E peekLast();

public abstract boolean removeFirstOccurrence(java.lang.Object);

public abstract boolean removeLastOccurrence(java.lang.Object);

=>The following are the implementation classes of Deque<E>:

(i)ArrayDeque<E> - Sequence

(ii)LinkedList<E> - NonSequence

Ex : DemoDeque.java

```
package maccess;
import java.util.*;
public class DemoDeque {
      @SuppressWarnings("removal")
```

```
    public static void main(String[] args) {
        Deque<Integer> ob = new ArrayDeque<Integer>();
        for(int i=1;i<=5;i++)
        {
          ob.add(new Integer(i));
        }//end of loop
        System.out.println(ob.toString());
        ob.addFirst(new Integer(11));
        ob.addLast(new Integer(12));
        System.out.println(ob.toString());
        ob.removeFirst();
        ob.removeLast();
        System.out.println(ob.toString());
        ob.pollFirst();
        ob.pollLast();
        System.out.println(ob.toString());
        System.out.println("First ele:"+ob.getFirst());
        System.out.println("Last ele:"+ob.getLast());
        System.out.println(ob.toString());
        System.out.println("peek first:"+ob.peekFirst());
        System.out.println("peek last:"+ob.peekLast());
        System.out.println(ob.toString());
        ob.addFirst(new Integer(11));
        ob.addFirst(new Integer(12));
        ob.addLast(new Integer(11));
        ob.addLast(new Integer(12));
        System.out.println(ob.toString());
        ob.removeFirstOccurrence(new Integer(11));
        System.out.println(ob.toString());
        ob.removeLastOccurrence(new Integer(12));
        System.out.println(ob.toString());

    }
}
```

o/p:

[1, 2, 3, 4, 5]

[11, 1, 2, 3, 4, 5, 12]

[1, 2, 3, 4, 5]

[2, 3, 4]

First ele:2

Last ele:4

[2, 3, 4]

peek first:2

peek last:4

[2, 3, 4]

[12, 11, 2, 3, 4, 11, 12]

[12, 2, 3, 4, 11, 12]

[12, 2, 3, 4, 11]

==========

Note:

=>LinkedList<E> is the implementation of both List<E> and Deque<E>.

==========

faq:

define Iterable<E>?

=>Iterable<E> is an interface from java.lang package and which provide the following methods to perform iterations of Collection<E> objects.

(i)iterator()

(ii)spliterator()

(iii)forEach()

(i)iterator():

=>iterator() method is used to create implementation object for Iterator<E> interface.

(ii)spliterator():

=>spliterator() method is used to create implementation object for Spliterator<T> interface.

(iii)forEach():

=>forEach() method introduced by Java8 version and which is used retrieve elements from Collection<E> objects directly.

==========

Note:

=>Iterable<E> is a Parent-Interface of Collection<>E>

==========

Limitation of Collection<E>:

=>In the process of organizing Database table data using Collection<E>,the Collection<E> cannot differentiate primary-key and NonPrimary-key-Values

Note:

This Limitation of Collection<E> can be overcomed using Map<K,V>

==========

faq:

define Map<K,V>?

=>Map<K,V> is an interface from java.util package and which organizes elements in the form of Key-Value pairs.

K - Key

V - Value

=>The following are some important methods of Map<K,V>:

```
public abstract int size();
public abstract boolean isEmpty();
public abstract boolean containsKey(java.lang.Object);
public abstract boolean containsValue(java.lang.Object);
public abstract V get(java.lang.Object);
public abstract V put(K, V);
public abstract V remove(java.lang.Object);
public abstract void putAll(java.util.Map<? extends K, ? extends V>);
public abstract void clear();
public abstract java.util.Set<K> keySet();
public abstract java.util.Collection<V> values();
public default void forEach
    (java.util.function.BiConsumer<? super K, ? super V>);
```

=>The following are the implementation classes of Map<K,V>:

(a)HashMap<K,V>

(b)LinkedHashMap<E>

(c)TreeMap<K,V>

(d)Hashtable<K,V>

(a)HashMap<K,V>:

=>HashMap<K,V> organizes elements without any order and which is NonSynchronized

class

(b)LinkedHashMap<E>:

=>LinkedHashMap<K,V> organizes elements in insertion order and which is also NonSynchronized class.

(c)TreeMap<K,V>:

=>TreeMap<K,V> organizes elements automatically in ascending order based on Primary Key and which is also NonSynchronized class.

(d)Hashtable<K,V>:

=>Hashtable<K,V> organizes elements without any order and which is synchronized class.

==========

Dt : 16/11/2022

Ex-program:

Note:

=>Construct one User defined class having the variables equal to the NonPrimary values of Database table.

EmployeeValues.java

```
package test;
public class EmployeeValues extends Object{
   public String name,desg;
   public int bSal;
```

```
    public float totSal;
    public EmployeeValues(String name,String desg,int bSal,float
totSal)
    {
          this.name=name;
          this.desg=desg;
          this.bSal=bSal;
          this.totSal=totSal;
    }
    @Override
    public String toString() {
          return name+"\t"+desg+"\t"+bSal+"\t"+totSal;
    }
}
```

DemoMap.java(MainClass)

```
package maccess;

import java.util.*;

import test.*;

public class DemoMap {

        public static void main(String[] args) {

     Scanner s = new Scanner(System.in);

     Map<String,EmployeeValues> ob=null;

     String name = null;

     try(s;){

        try {

                while(true) {

                        System.out.println("****Choice****");



System.out.println("1.HashMap\n2.LinkedHashMap\n3.TreeMap\n4.Hashtable\n5.exit");

                        System.out.println("Enter the Choice:");
```

```
switch(Integer.parseInt(s.nextLine())) {
case 1:
        ob = new HashMap<String,EmployeeValues>();
        name="HashMap";
        break;
case 2:
        ob = new LinkedHashMap<String,EmployeeValues>();
        name="LinkedHashMap";
        break;
case 3:
        ob = new TreeMap<String,EmployeeValues>();
        name="TreeMap";
        break;
case 4:
        ob = new Hashtable<String,EmployeeValues>();
        name="Hashtable";
        break;
case 5:
        System.out.println("Operations Stopped on Map...");
        System.exit(0);
default:
        System.out.println("Invalid Choice...");
}//end of switch
System.out.println("perform operations on "+name);
```

```
xyz:
while(true){
        System.out.println("====Choice====");

System.out.println("1.put(K,V)\n2.remove(object)\n3.get(object)\n4.keySet()\n5.values()\n6.
exit");
        System.out.println("Enter the Choice:");
        switch(Integer.parseInt(s.nextLine())) {
        case 1:
            System.out.println("Enter the empId:");
            String id = s.nextLine();
            System.out.println("Enter the empName:");
            String eName = s.nextLine();
            System.out.println("Enter the Desg:");
            String desg = s.nextLine();
            System.out.println("Enter the bSal:");
            int bSal = Integer.parseInt(s.nextLine());
            float totSal = bSal+(0.93F*bSal)+(0.63F*bSal);
            ob.put(new String(id),
              new EmployeeValues(eName,desg,bSal,totSal));
            ob.forEach((p,q)->
            {
                System.out.println(p+"\t"+q);
            });
```

```
                break;
        case 2:
                if(ob.isEmpty()) {
                        System.out.println("Map is empty...");
                }else {
                        System.out.println("Enter empId:");
                        String eId = new String(s.nextLine());
                        if(ob.containsKey(eId)) {
                                ob.remove(eId);
                                System.out.println("Details removed...");
                                ob.forEach((p,q)->
                        {
                                System.out.println(p+"\t"+q);
                        });
                        }else {
                                System.out.println("Invalid eId...");
                        }
                }
                break;
        case 3:
                if(ob.isEmpty()) {
                        System.out.println("Map is empty...");
                }else {
                        System.out.println("Enter empId:");
```

```
                String eId = new String(s.nextLine());
                if(ob.containsKey(eId)) {
                        EmployeeValues ev = ob.get(eId);
            System.out.println(ev.toString());
                }else {
                        System.out.println("Invalid eId...");
                }
            }
            break;
        case 4:
            if(ob.isEmpty()) {
                System.out.println("Map is empty...");
            }else {
              Set<String> ob2 = ob.keySet();
              ob2.forEach((z)->
              {
                  System.out.println(z.toString());
               });
            }
            break;
        case 5:
            if(ob.isEmpty()) {
                System.out.println("Map is empty...");
            }else {
```

```
                    Collection<EmployeeValues> ob3 = ob.values();
                    ob3.forEach((z)->
                    {
                            System.out.println(z.toString());
                    });
                }
                    break;
            case 6:
                    System.out.println("Operations stopped on "+name);
                    break xyz;
            default:
                    System.out.println("Invalid Choice...");
                    }//end of switch
                }//end of loop
            }//end of loop
        }catch(Exception e) {e.printStackTrace();}
    }//end of try
        }
}
```

o/p:

```
****Choice****
1.HashMap
2.LinkedHashMap
3.TreeMap
```

4.Hashtable

5.exit

Enter the Choice:

3

perform operations on TreeMap

====Choice====

1.put(K,V)

2.remove(object)

3.get(object)

4.keySet()

5.values()

6.exit

Enter the Choice:

1

Enter the empId:

A121

Enter the empName:

Raj

Enter the Desg:

SE

Enter the bSal:

16000

A121 Raj SE 16000 40960.0

====Choice====

1.put(K,V)

2.remove(object)

3.get(object)

4.keySet()

5.values()

6.exit

Enter the Choice:

1

Enter the empId:

A001

Enter the empName:

Ram

Enter the Desg:

TE

Enter the bSal:

15000

A001 Ram TE 15000 38400.0

A121 Raj SE 16000 40960.0

====Choice====

1.put(K,V)

2.remove(object)

3.get(object)

4.keySet()

5.values()

6.exit

Enter the Choice:

1

Enter the empId:

A021

Enter the empName:

Alex

Enter the Desg:

ME

Enter the bSal:

14000

A001 Ram TE 15000 38400.0

A021 Alex ME 14000 35840.0

A121 Raj SE 16000 40960.0

====Choice====

1.put(K,V)

2.remove(object)

3.get(object)

4.keySet()

5.values()

6.exit

Enter the Choice:

4

A001

A021

A121

====Choice====

1.put(K,V)

2.remove(object)

3.get(object)

4.keySet()

5.values()

6.exit

Enter the Choice:

5

Ram TE 15000 38400.0

Alex ME 14000 35840.0

Raj SE 16000 40960.0

====Choice====

1.put(K,V)

2.remove(object)

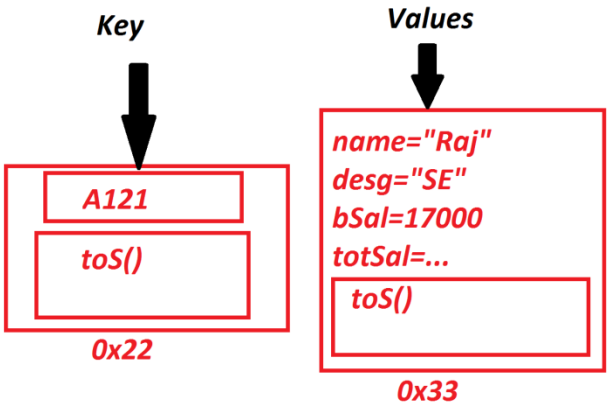
3.get(object)

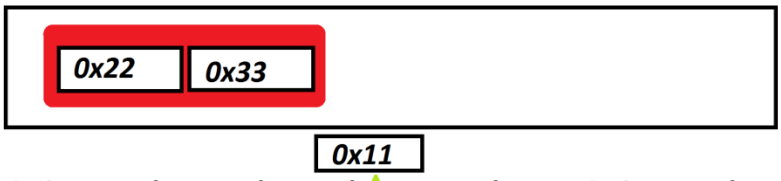
4.keySet()

5.values()

6.exit

Enter the Choice:





Map<String,EmployeeValues> ob=new HashMap<String,EmployeeValues>();

This ref will hold Unlimited String-EmployeeValues objects pairs

==========

**imp*

Cursor Statements in JCF:

=>The statements which are used to retrieve elements from Collection<E> objects are known as Cursor Statements.

=>The following are some important Cursor statements in JCF:

(a)Iterator<E> ----> Collection<E> objects

(b)ListListerator<E>----> List<E> objects

(c)Enumeration<E> ----> Vector<E> objects

(d)Spliterator<T> ----> Array Objects and Collection<E> objects

(a)Iterator<E>:

=>Iterator<E> is an interface from java.util package and which is used to retrieve elements from Collection<E> objects in forward direction.

syntax:

Iterator<E> it = obj.iterator();

(b)ListIterator<E>:

=>ListIterator<E> is also an interface from java.util package and which is used to retrieve elements from List<E> objects in both directions forward and backward.

=>ListIterator<E> is a Child-Interface of Iterator<E>.

=>The following are some important methods of ListIterator<E>:

```
public abstract boolean hasNext();
public abstract E next();
public abstract boolean hasPrevious();
public abstract E previous();
```

(c)Enumeration<E>:

=>Enumeration<E> is an interface from java.util package and which is used to retrieve elements from Vector<E> objects.

=>The following are some important methods of Enumeration<E>:

```
public abstract boolean hasMoreElements();
public abstract E nextElement();
```

(d)Spliterator<T>:

=>Spliterator<T> interface introduced by Java8 version and which is used to retrieve elements from Arrays objects and Collection<E> objects.

==========

Dt : 17/11/2022

Ex-program : DemoCursorStatements.java

```
package maccess;
import java.util.*;
public class DemoCursorStatements {
      public static void main(String[] args) {
        Vector<Integer> v = new Vector<Integer>();
        for(int i=1;i<=10;i++)
        {
         v.add(new Integer(i));
        }//end of loop
        System.out.println("*****ListIterator<T>****");
        ListIterator<Integer> li = v.listIterator();
         //creating implementation object for ListIterator<T>
         //This object will hold the reference of List<E> object
        System.out.print("Forward : ");
        while(li.hasNext()) {
         System.out.print(li.next()+" ");
        }//end of loop
        System.out.print("\nBackward : ");
        while(li.hasPrevious()) {
         System.out.print(li.previous()+" ");
        }//end of loop
        System.out.println("\n****Enumeration<E>*****");
        Enumeration<Integer> e = v.elements();
      //creating implementation object for Enumeration<T>
        //This object will hold the reference of Vector<E> object
        while(e.hasMoreElements()) {
         System.out.print(e.nextElement()+" ");
        }//end of loop
      }
}
```

o/p:

*****ListIterator<T>****

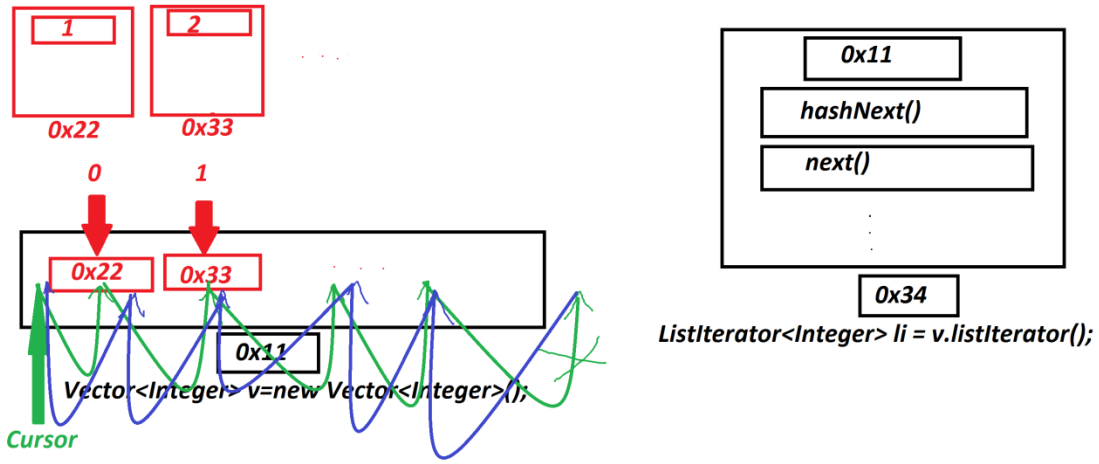
Forward : 1 2 3 4 5 6 7 8 9 10

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

****Enumeration<E>*****

1 2 3 4 5 6 7 8 9 10

diagram:



==========

faq:

define forEach() method?

=>forEach() method introduced by Java8 version and which is used to retrieve elements from Collection<E> and Map<K,V> objects

Method Signature of forEach() on Collection<E>:

public default void forEach(java.util.function.Consumer<? super T>);

Method Signature of forEach() on Map<K,V>:

public default void forEach

(java.util.function.BiConsumer<? super K, ? super V>);

faq:

define Consumer<T>?

=>Consumer<T> is a functional interface introduced by Java8 version and which provide abstract method "accept(T)" to hold LambdaExpression passed as parameter to forEach() method on Collection<E> objects.

structure of Consumer<T>:

```
public interface java.util.function.Consumer<T>
{
  public abstract void accept(T);
}
```

faq:

define BiConsumer<T,U>?

=>BiConsumer<T,U> is a functional interface introduced by Java8 version and which provide abstract method "accept(T,U)" to hold LambdaExpression passed as parameter to forEach() method on MapK,V> objects.

structure of BiConsumer<T,U>:

public interface java.util.function.BiConsumer<T, U>

{

public abstract void accept(T, U);

}

==========

****imp***

define Enum<E>?

=>Enum<E> is a abstract class from java.lang package.

=>we use "enum" keyword to create implementation objects for Enum<E>.

syntax:

enum Enum_name

{

//elements

//variables

//methods

}

=>Enum<E> is a collection of elements,variables,constructors and methods.

=>The constructors which are declared within the Enum<E> are automatically

private constructors.

EX-program:

Cars.java

```
package test;
public enum Cars {
    alto(1200),dezire(1400),figo(1600);
      public int price;
      private Cars(int price)
      {
            this.price=price;
      }
      public int getPrice() {
            return price;
      }
      public void setPrice(int price) {
            this.price = price;
      }

}
```

DemoEnum.java(MainClass)

```
package maccess;

import test.Cars;

import java.util.Scanner;

public class DemoEnum {

        public static void main(String[] args) {

          Scanner s = new Scanner(System.in);

    Cars c[] = Cars.values();

    for(Cars k : c)

    {

   System.out.println(k.toString()+" Costs "+k.price+" thousand dollars");

    }//end of loop

    System.out.println("====set data using setter methods===");

    for(Cars p : c)

    {
```

```
            System.out.println("Enter the price for "+p.toString());
            p.setPrice(s.nextInt());
        }//end of loop
        System.out.println("====get data using getter methods====");
        for(Cars q : c)
        {
 System.out.println(q.toString()+" Costs "+q.getPrice()+" thousand dollars");
        }//end of loop
        s.close();
            }
}
```

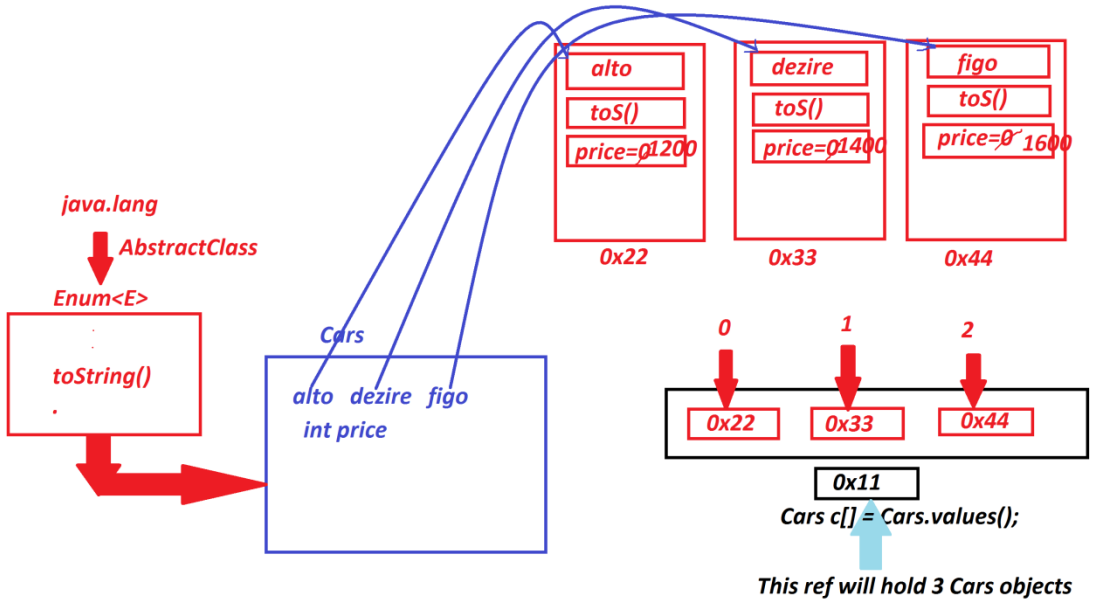
o/p:

```
alto Costs 1200 thousand dollars
dezire Costs 1400 thousand dollars
figo Costs 1600 thousand dollars
====set data using setter methods===
Enter the price for alto
1700
Enter the price for dezire
2300
Enter the price for figo
1700
====get data using getter methods====
alto Costs 1700 thousand dollars
```

dezire Costs 2300 thousand dollars

figo Costs 1700 thousand dollars

diagram:



==========

Note:

=>In realtime Enum<E> is used in application to hold defined list of elements.

Ex:

Week Days

Months in a Year

==========

Dt: 18/11/2022

List of Objects Generated from CoreJava:

1.User defined class objects

2.String Objects

3.WrapperClass Objects

4.Array Objects

5.Collection<E> objects

6.Map<K,V< objects

7.Enum<E> objects

Complete List of Objects:

1.User defined class objects

2.String Objects

(a)String class Objects

(b)StringBuffer class Objects

(c)StringBuilder class Objects

3.WrapperClass Objects

(a)Byte Objects

(b)Short Objects

(c)Integer Objects

(d)Long Objects

(e)Float Objects

(f)Double Objects

(g)Character Objects

(h)Boolean Objects

4.Array Objects

(a)Array holdding User defined class Objects

(b)Array holding String Objects

(c)Array holding WrapperClass Objects

(d)Array holding Array Objects(Jagged Arrays)

(e)Array holding Dis-Similer Objects(Object Array)

5.Collection<E> objects

(a)Set<E>

(i)HashSet<E> Objects

(ii)LinkedHashSet<E> Objects

(iii)TreeSet<E> Objects

(b)List<E>

(i)ArrayList<E> Objects

(ii)LinkedList<E> Objects

(iii)Vector<E> Objects

=>Stack<E> Objects

(c)Queue<E>

=>PriorityQueue<E> Objects

(d)Deque<E>

(i)ArrayDeque<E> Objects

(ii)LinkedList<E> Objects

6.Map<K,V< objects

(a)HahMap<K,V> Objects

(b)LinkedHashMap<K,V> Objects

(c)TreeMap<K,V> Objects

(d)Hashtable<K,V> Objects

7.Enum<E> objects

==========

faq:

wt is the diff b/w

(a)Container Objects

(b)Utility Objects

(c)Cursor Objects

(a)Container Objects:

=>The Objects which hold data are known as Container Objects.

(b)Utility Objects:

=>The Objects which perform operations on other objects are known as Utility Objects.

Ex:

Scanner

StringTokenizer

StringJoiner

Arrays

Collections

(c)Cursor Objects:

=>The Objects which are used to retrieve data from Collection objects are known as Cursor Objects.

Ex:

Iterator<E>

ListIterator<E>

Enumeration<E>

Spliterator<T>

==========

faq:

wt is the diff b/w

(a)Collection<E>

(b)Collections

(a)Collection<E>:

=>Collection<E> is an interface from java.util package and which is root of Java Collection<E> Framework

(b)Collections:

=>"Collections" is a utility class from java.util package and provide the following methods to perform operations on Collection<E> objects

sort()

binarySearch()

==========

**imp*

Multi-Threading process in Java:

define Application?

=>set-of-programs collected together to perform defined action is known as Application.

define process?

=>The application under execution is a process.(According Java)

define Task?

=>The part of process is known as Task.

Note:

=>According to Java Application,each program in application is a Task.

define Multi-Tasking?

=>Executing multiple tasks simultaneously is known as Multi-Tasking. (Simultaneously means at-a-time but not parallel)

Note:

=>In the process of executing multiple tasks only some part of task is executed known as Thread.

define Thread?

=>The part of task is known as Thread.

=>Thread is a LightWeight and Background process.

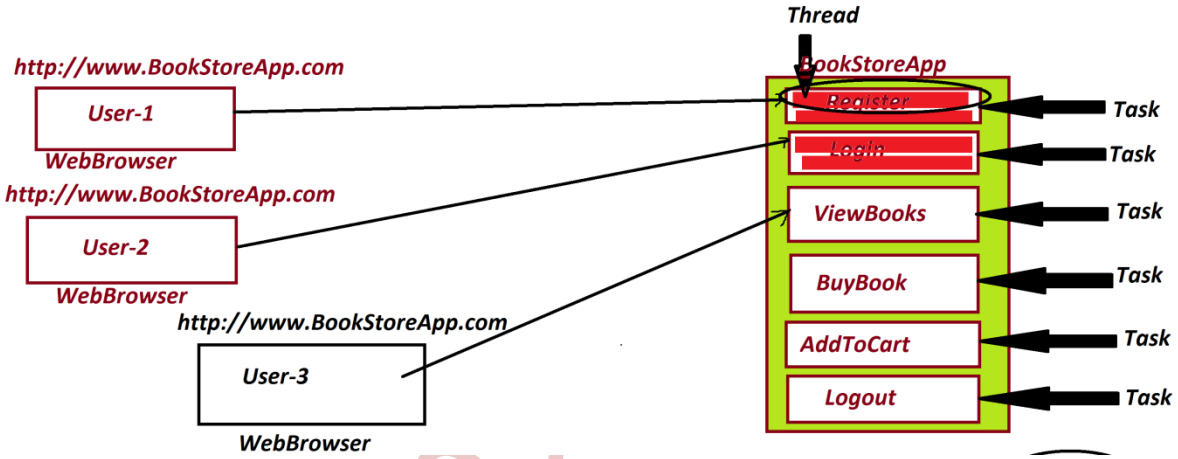
LightWeight process - means consumes less execution time.

Background process - means there is no separate identification.

define Multi-threading?

=>Executing muliple threads simultaneously is known as Multi-Threading.

Diagram:



==========

**imp*

creating and Execution Threads:

step-1 : The user defined class must be implemented from "java.lang.Runnable" interface

Structure of Runnable Interface:

```
public interface java.lang.Runnable
{
   public abstract void run();
```

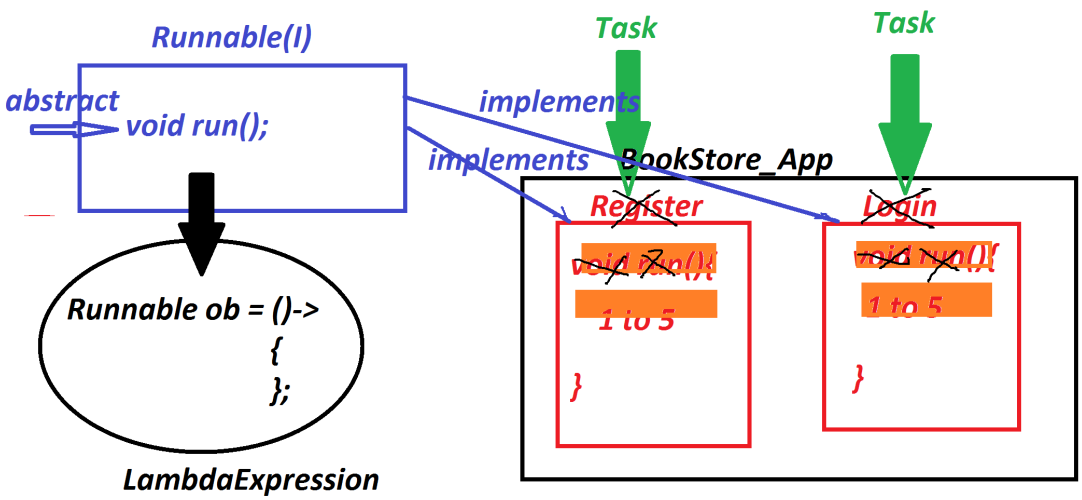
}

step-2 : The user defined implementation class must construct body for "run()" method and which is holding program-logic

step-3 : Create object for User defined implementation class

step-4 : create object for pre-defined "Thread" class and while object creation we pass User defined implementation class object-reference as parameter

step-5 : execute run() method using start() method.



Ex:

Register.java

```
package test;
public class Register implements Runnable{
	@Override
```

```
    public void run() {
     for(int i=1;i<=5;i++) {
          System.out.println("Registration for
"+Thread.currentThread().getName());
          try {
               Thread.sleep(2000);
          }catch(Exception e) {e.printStackTrace();}
     }
    }
}
```

Login.java

```
package test;
public class Login implements Runnable{
     @Override
   public void run(){
        for(int i=1;i<=5;i++) {
          System.out.println("Login for
"+Thread.currentThread().getName());
          try {
          Thread.sleep(2000);
          }catch(Exception e) {e.printStackTrace();}
     }
   }
}
```

DemoThread1.java(MainClass)

```
package maccess;
import test.*;
public class DemoThread1 {
     public static void main(String[] args) {
        Register ob1 = new Register();
        Login ob2 = new Login();

        Thread t1 = new Thread(ob1);
        Thread t2 = new Thread(ob2);

        t1.setName("User-1");
        t2.setName("User-2");

        t1.setPriority(Thread.MAX_PRIORITY-2);//8
        t2.setPriority(Thread.MAX_PRIORITY-1);//9
        t1.start();
```

```
        t2.start();

        System.out.println("Min Priority :
"+Thread.MIN_PRIORITY);
        System.out.println("Max Priority :
"+Thread.MAX_PRIORITY);
        System.out.println("Normal Priority :
"+Thread.NORM_PRIORITY);
     }
}
```

o/p:

Registration for User-1

Login for User-2

Registration for User-1

Login for User-2

Registration for User-1

Login for User-2

Registration for User-1

Login for User-2

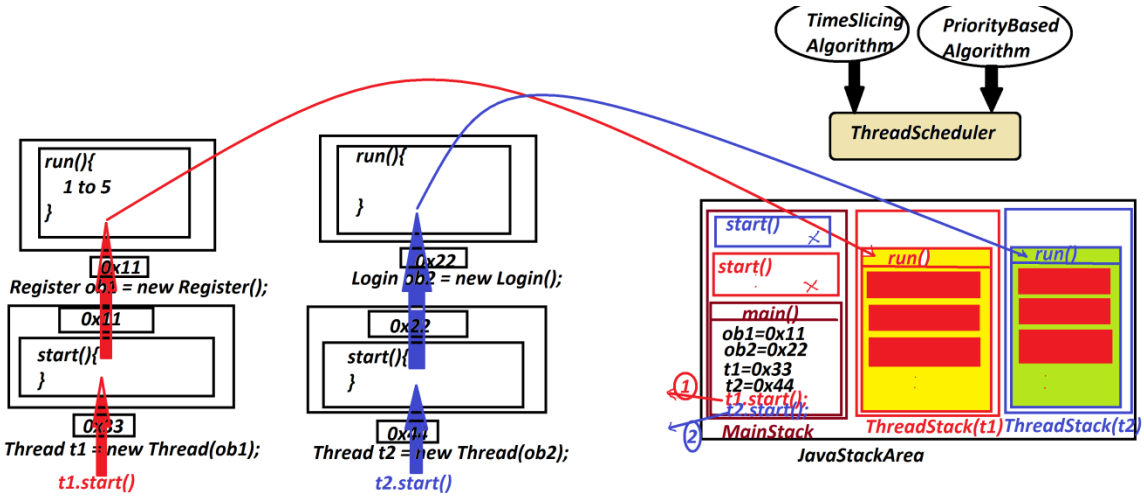
Registration for User-1

Login for User-2

==========

Dt : 19/11/2022

Execution flow of above program:



==========

Note:

=>In the process of executing Multiple threads,Multiple thread-stacks are created and all these Multiple thread-stacks are executed Simultaneously.

==========

faq:

define start() method?

=>start() is a pre-defined method from java.lang.Thread class and which is used to create new thread for execution.

(i)start() method specify to create separate thread-stack

(ii)stack() method will activate Thread-Scheduler.

(iii)start() will load run() method onto thread-stack

==========

faq:

define Thread Scheduler?

=>Thread Scheduler is a pre-defined algorithm to control and manage threads

for execution.

=>Thread Scheduler will use the following algorithms:

(a)Time-Slicing Algorthm

(b)Priority based Algorithm

(a)Time-Slicing Algorthm:

=>In Time-Slicing algorithm all the multiple threads are executed based on defined time-slice.

=>Time-Slicing algorithm is default algorithm used by Thread-Scheduler.

(b)Priority based Algorithm:

=>In Priority Based algorithm the threads are executed based on thread priorities.

=>The following fields from java.lang.Thread class represent priorities:

```
public static final int MIN_PRIORITY;
public static final int NORM_PRIORITY;
public static final int MAX_PRIORITY;
```

=>we use setPriority() method to set priority for threads:

syntax:

```
t1.setPriority(Thread.MAX_PRIORITY-2);
t2.setPriority(Thread.MAX_PRIORITY-1);
```

==========

***

imp

Creating Thread using LambdaExpression:(Java8)

=>In LambadaExpression process the run() method is declared without name,

which means as Anonymous method or LambdaExpression.

Ex : DemoThread2.java

```
package maccess;
public class DemoThread2 {
     public static void main(String[] args) {

         new Thread(()->
         {
           for(int i=1;i<=5;i++) {
                System.out.println("Registration for
"+Thread.currentThread().getName());
                try {
                     Thread.sleep(2000);
                }catch(Exception e) {e.printStackTrace();}
           }
         }).start();

         new Thread(()->
         {
             for(int i=1;i<=5;i++) {
                System.out.println("Login for
"+Thread.currentThread().getName());
                try {
                Thread.sleep(2000);
                }catch(Exception e) {e.printStackTrace();}
           }
        }).start();

         new Thread(()->
         {
             for(int i=1;i<=5;i++) {
                System.out.println("View Books for
"+Thread.currentThread().getName());
                try {
                Thread.sleep(2000);
                }catch(Exception e) {e.printStackTrace();}
           }
        }).start();
  }
```

```
}
```

```
==========================================================================
```