WHY WE ARE LEARNING JAVA 1.8?wHY IT IS INTRODUCED ?

* 1995 java 1.0
* 2012-2013 black period for java language .Started to migrate to other language leading to decrease value of java
* Reasons are ,
* Lengthy boiler plate code (number of lines of code)
* Requires more time for development
* To overcome this problem they java language creators introduced java 1.8 (march 18,2014)
* Version which reduce number of lines of code. Version incorporated several concept to achieve code conciseness
* Due to this java survived and again people shifted back to java language because features or concepts introduced in java 1.8
* Answer: Most of the java project uses java 1.8 concept for code conciseness. without using java 1.8 code duplication percentage will be reduced.

===================================================================================================================================

*SONARQUBE TOOL:TOOL IS USED TO CHEQUE CODE QUALITY*

* *duplicate code (10 to 70 % code is acceptable)*
* *check rules and conventions*
* *check test cases(min 80% acceptable)*

*while doing project it is mandatory to use/implement java 8 features to reduce code duplication.*

===================================================================================================================================

FEATURES INTRODUCED IN JAVA 1.8:

* lambda expression
* functional interface
* predefined functional interface
* static and default methods
* :: operator (method reference)
* <> diamond operator
* constructor reference
* streams
* date and time
* optional class ….*<other features>*

===================================================================================================================================

CONCEPT 01:LAMBADA EXPRESSION:

* Anonymous function
* It does not have modifiers
* It does not have identifier/name
* It does not have return type
* To hold lambda expression we need a functional interface
* The main objective of lambda of expression is enabling functional programing in java
* Lisp is programing language where they used lambda expression for the first time
* {functional programing is enabled with lambda expression released in java 1.8}
* Functional interface=interface having single abstract method
* '@FunctionalInterface' annotation is recommended to use
* HOW CODE IS REDUCED?
* While writing lambda expression writing datatypes are optional
* Parenthesis and curly braces are optional in case of single arument and statement respectively otherwise it is compulsory
* When there is single return statement ,should not write return statement otherwise will get error
* Lambda expression is one of the way to implement functional interface
* Number of argument in lambda expression and abstract method of functional interface must be equal

|  |  |  |  |
| --- | --- | --- | --- |
| (optional:formal arguments)-->{  //stmt  }; | ()-->{  SYSO("THIS IS MSG")  }; | ()-->SYSO("THIS IS MSG"); | -->SYSO("THIS IS MSG"); |
| (int a)-->{return a\*a} | ( a)-->{return a\*a} | a-->{return a\*a} | a--> a\*a |
| ( a,b)-->{return a\*b} | ( a,b)-->SYSO(a\*b) |  |  |

===================================================================================================================================

|  |
| --- |
|  |
| public class Driver {  public static void main(String[] args) {  // overiding method in functional interface usinf=g lambda expression  checkNumberEvenOdd var1 = (a) -> {  return (a % 2 == 0) ? true : false;  };  System.out.println(var1.checkNum(7));  // --------------------------------------------------------------------------------------  // no compulsion to use parenthesis when there is single formal argument  // no compulsion to use curly braces when there is single statement  // implesit return type  checkNumberEvenOdd var2 = a -> (a % 2 == 0) ? true : false;  System.out.println(var2.checkNum(6));  }  }  interface checkNumberEvenOdd {  boolean checkNum(int n);  } |
|  |

|  |
| --- |
|  |
| public class Driver3 {  public static void main(String[] args) {  Voter var1= a-> (a>18)?"is eligible for voating":"not eligible for voating";  System.out.println(var1.printVoterOrNot(19));  System.out.println(var1.printVoterOrNot(15));  Discount var2=a-> (a>60)?"Your are eligible for discount":"Your are not eligible for discount";  System.out.println(var2.printDiscountMsg(99));  System.out.println(var2.printDiscountMsg(15));  Authorization var3=a-> (a.toLowerCase().equals("admin"))?"You are Authorized..!":"You are not Authorized..!";  System.out.println(var3.printAuthorizationMsg("Admin"));  System.out.println(var3.printAuthorizationMsg("team lead"));  }  }  @FunctionalInterface  interface Voter {  String printVoterOrNot(int age);  }  @FunctionalInterface  interface Discount {  String printDiscountMsg(int age);  }  @FunctionalInterface  interface Authorization {  String printAuthorizationMsg(String roleName );  } |
| is eligible for voating  not eligible for voating  Your are eligible for discount  Your are not eligible for discount  You are Authorized..!  You are not Authorized..! |

*AISSIGNMENT ON LAMBDA EXPRESSION: give implementation for comparable ,comparator and runnable interface using lambda expression*

|  |
| --- |
| ***assignment on static methods-TASK1/2*** |
| import java.util.ArrayList;  import java.util.Collection;  import java.util.Collections;  import java.util.List;  public class Driver4AssignmentComparableComparator {  public static void main(String[] args) {  List<UserInfo> userInfoList=new ArrayList<>();  userInfoList.add(new UserInfo("AB-name1", 123));  userInfoList.add(new UserInfo("AA-name1", 456));  userInfoList.add(new UserInfo("C-name1", 789));  userInfoList.add(new UserInfo("D-name1", 126));  userInfoList.forEach(System.out::println);  System.out.println("------sorting using comparable-------------------------------------");    Collections.sort(userInfoList);  userInfoList.forEach(System.out::println);  System.out.println("----------sorting using comparator---------------------------------");  userInfoList.sort((o1,o2)->o2.name.compareTo(o1.name));  userInfoList.forEach(System.out::println);    }  }  class UserInfo implements Comparable<UserInfo>{  String name;  int id;  public UserInfo(String name, int id) {  this.name = name;  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  //lambda expression to sort object in desending order..!  Comparable<UserInfo> comparableObject = o -> o.name.compareTo(this.name);  @Override  public int compareTo(UserInfo o) {  return comparableObject.compareTo(o);  }  @Override  public String toString() {  return "id :"+id+" name :"+name;  }  } |
| id :123 name :AB-name1  id :456 name :AA-name1  id :789 name :C-name1  id :126 name :D-name1  ------sorting using comparable-------------------------------------  id :126 name :D-name1  id :789 name :C-name1  id :123 name :AB-name1  id :456 name :AA-name1  ----------sorting using comparator---------------------------------  id :126 name :D-name1  id :789 name :C-name1  id :123 name :AB-name1  id :456 name :AA-name1 |

|  |
| --- |
| ***assignment on lambda expression-TASK2/2*** |
| public class Driver4Assignment1Task {  public static void main(String[] args) {  Runnable thread1= ()-> {  for (int i = 0; i < 3; i++) {  System.out.println("thread 1..!");  }  };  Thread t1=new Thread(thread1);  t1.start();  Runnable thread2=()-> {  for (int i = 0; i < 3; i++) {  System.out.println("thread 2..!");  }  };  Thread t2=new Thread(thread2);  t2.start();  }  } |
| thread 1..!  thread 1..!  thread 2..!  thread 2..!  thread 2..!  thread 1..! |

FUNCTIONAL INTERFACE:

* Interface having single an abstract method is called as functional interface
* functional interface=SAM
* Feature introduced in java 1.8
* To hold lambda expression we need functional interface reference variable
* '@Functionalinterface' annotation is used to specify the functional interface
* Inside functional interface we can have object class method as abstract method
* We can have tostring(),equals() and hashcode() methods as abstract method
* Why other are not abstract method
* clone is native method
* finalize() depreciated method
* other are final , inside a functional interface we have object class method as abstract methods ,inside object class we have 11 methods,
* 01) `public String toString()`
* 02) `public boolean equals(Object obj)`
* 03) `public int hashCode()`
* -----------------------------------------------------------------------------------------
* 04) `protected Object clone() throws CloneNotSupportedException`
* -----------------------------------------------------------------------------------------
* 05) `protected void finalize() throws Throwable`
* -----------------------------------------------------------------------------------------
* 06) `public final Class<?> getClass()`
* 07) `public final void notify()`
* 08) `public final void notifyAll()`
* 09) `public final void wait() throws InterruptedException`
* 10) `public final void wait(long timeout) throws InterruptedException`
* 11) `public final void wait(long timeout, int nanos) throws InterruptedException`

|  |
| --- |
| ***Program for checking scenarios of functional interface*** |
| // package javapgms.oops.functionalinterface;  public class Driver1 {  public static void main(String[] args) {    }  }   |  |  |  |  | | --- | --- | --- | --- | | @FunctionalInterface  interface Demo1{  void m1();  } | @FunctionalInterface  interface Demo2{  void m1();  boolean equals(Object o);  } | // @FunctionalInterface  // interface Demo3{  // void m1();  // void m2();  // } | @FunctionalInterface  interface Demo4{  void m1();  String toString();  } | | @FunctionalInterface  interface Demo5{  void m1();  int hashCode();  } | // @FunctionalInterface  // interface Demo6{  // void m1();  // void finalize();  // } | // @FunctionalInterface  // interface Demo7{  // void m1();  // void notify();  // } | // @FunctionalInterface  // interface Demo8{  // void m1();  // Object clone();//as this is prefixed with protected  // } | | // @FunctionalInterface  // interface Demo9{  // void m1();  // Class getClass();  // } | // @FunctionalInterface  // interface Demo10{  // void m1();  // int hashCode();  // String toString();  // boolean equals();  // } |  |  | |
|  |

|  |
| --- |
| ***print collection of treeset in descending order*** |
| public TreeSet(Comparator c){  //statement  }  PQ:chang the default sorting (asending order) of treeset collection to descending order |
| import java.util.Comparator;  import java.util.TreeSet;  public class Driver4 {  public static void main(String[] args) {  Comparator<Integer> comparatorObject = (o1, o2) -> o2.compareTo(o1);  TreeSet<Integer> treesetlist = new TreeSet<>(comparatorObject);  treesetlist.add(77);  treesetlist.add(9);  treesetlist.add(67);  treesetlist.add(48);  treesetlist.add(57);  System.out  .println("changing default ordering of treeset collection(desending..!)-----------------------------");  System.out.println(treesetlist);  }  } |
| [77, 67, 57, 48, 9] |

STATIC METHODS IN INTERFACE :

* It is introduced in jdk 1.8
* The main purpose of introducing static methods in interface to reduce number of lines of code to
* increase the code readability and remove the duplicate code also
* we can call this method using method signature and interface name as reference
* static methods can't be inherited the child class
* the implementation which is common for all the Childs is written in static method in interface
* we can have more than one static method inside interface
* we can have main method inside a interface

|  |
| --- |
| ***assignment on static methods-TASK1/2*** |
| // package javapgms.java8.practicals.StaticMethods\_02;  public class Driver1 {  public static void main(String[] args) {  Car1 ob1 = new Car1();  Bicycal ob2 = new Bicycal();  ob1.fuel();  ob2.fuel();  Vehical.commonActivity();  }  }  interface Vehical {  void fuel();  static void commonActivity() {  start();  run();  stop();  }  static void start() {  System.out.print("START ");  }  static void run() {  System.out.print("RUN ");  }  static void stop() {  System.out.print("STOP ");  }  }  class Car1 implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption:disel/petrol");  }  }  class Bicycal implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption: No fuel");  }  } |
| [COMMON FEATURES ] :calling | SMS | FM  [SMARTPHONES] : video call | play games |install and use apps | can run browser | camera  [TABLETS ] : can VIDEO games | can code |

|  |
| --- |
| ***assignment on static methods-TASK2/2*** |
| // package javapgms.java8.practicals.StaticMethods\_02;  public class Driver1 {  public static void main(String[] args) {  Car1 ob1 = new Car1();  Bicycal ob2 = new Bicycal();  ob1.fuel();  ob2.fuel();  Vehical.commonActivity();  }  }  interface Vehical {  void fuel();  static void commonActivity() {  start();  run();  stop();  }  static void start() {  System.out.print("START ");  }  static void run() {  System.out.print("RUN ");  }  static void stop() {  System.out.print("STOP ");  }  }  class Car1 implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption:disel/petrol");  }  }  class Bicycal implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption: No fuel");  }  } |
| fuel consumption:disel/petrol  fuel consumption: No fuel  START RUN STOP |

Questions and notes:

|  |
| --- |
|  |
| -Runnable :  -Comparable and Comparator are predefined functional interface used for used for sorting of custom object  -functional interface having only one method i.e run() method  -prototype  -present in java.lang package  -introduced in jdk 1.0 |
| what are native methods |
| comparable vs comparator |
| ways to implement functional interface |
| how to resolve problem/error occured while creating implementation class of interface?{T} |
| "in lambda expression no need to specify datatype of formal argument" why? |
|  |