**Java Learning Path**

**(**Java SE 8 Programmer II**)**

Table of Contents

[**1. Java Class Design** 4](#_Toc536717716)

[**1.1 Implement encapsulation** 4](#_Toc536717717)

[**1.2 Implement inheritance including visibility modifiers and composition** 5](#_Toc536717718)

[**1.3 Implement polymorphism** 5](#_Toc536717719)

[**1.4 Override hashCode, equals, and toString methods from Object class** 6](#_Toc536717720)

[**1.5 Create and use singleton classes and immutable classes** 8](#_Toc536717721)

[**1.5 Develop code that uses static keyword on initialize blocks, variables, methods, and classes** 10](#_Toc536717722)

[**2. Generics and Collections** 11](#_Toc536717723)

[**2.1 Create and use a generic class** 11](#_Toc536717724)

[**2.2 Create and use ArrayList, TreeSet, TreeMap, and ArrayDeque objects** 11](#_Toc536717725)

[ArrayList: 11](#_Toc536717726)

[TreeSet: 11](#_Toc536717727)

[TreeMap: 11](#_Toc536717728)

[ArrayDeque: 12](#_Toc536717729)

[**2.3 Use java.util.Comparator and java.lang.Comparable interfaces** 14](#_Toc536717730)

[Comparator: 14](#_Toc536717731)

[Comparable: 16](#_Toc536717732)

[**4. Java Stream API** 18](#_Toc536717733)

[**4.1 Create and manage date-based and time-based events including a combination of date and time into a single object using LocalDate, LocalTime, LocalDateTime, Instant, Period, and Duratio**n 18](#_Toc536717734)

[**4.2 Work with dates and times across time zones and manage changes resulting from daylight savings including Format date and times values** 20](#_Toc536717735)

[**4.3 Define and create and manage date-based and time-based events using Instant, Period, Duration, and TemporalUnit** 20](#_Toc536717736)

[**5. Java File I/O (NIO.2)** 20](#_Toc536717737)

[**5.1 Use Path interface to operate on file and directory paths** 20](#_Toc536717738)

[**5.2 Use Files class to check, read, delete, copy, move and manage metadata of a file or directory** 20](#_Toc536717739)

[**6. Advanced Java Class Design** 21](#_Toc536717740)

[**6.1 Develop code that uses abstract classes and methods** 21](#_Toc536717741)

[**6.2 Develop code that uses the final keyword** 22](#_Toc536717742)

[**6.3 Create inner classes including static inner class, local class, nested class, and anonymous inner class** 24](#_Toc536717743)

[**6.4 Use enumerated types including methods, and constructors in an enum type** 25](#_Toc536717744)

[**6.5 Develop code that declares, implements and/or extends interfaces and use the @Override annotation** 26](#_Toc536717745)

[**7. Exceptions and Assertions** 27](#_Toc536717746)

[**7.1 Use try-catch and throw statements** 27](#_Toc536717747)

[**7.2 Use catch, multi-catch, and finally clauses** 28](#_Toc536717748)

[**7.3 Use Autoclose resources with a try-with-resources statement** 29](#_Toc536717749)

[**7.4 Create custom exceptions and Auto-closeable resources** 29](#_Toc536717750)

[**7.5 Test invariants by using assertions** 30](#_Toc536717751)

[**8. Java I/O Fundamentals** 30](#_Toc536717752)

[**8.1 Read and write data from the console** 30](#_Toc536717753)

[**8.2 Use BufferedReader, BufferedWriter, File, FileReader, FileWriter, FileInputStream, FileOutputStream, ObjectOutputStream, ObjectInputStream, and PrintWriter in the java.io package.** 30](#_Toc536717754)

[**9. Java Concurrency** 32](#_Toc536717755)

[**9.1 Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks** 32](#_Toc536717756)

[**9.2 Identify potential threading problems among deadlock, starvation, livelock, and race conditions** 34](#_Toc536717757)

[Deadlock: 34](#_Toc536717758)

[Starvation Lock: 34](#_Toc536717759)

[Livelock: 34](#_Toc536717760)

[Race condition: 34](#_Toc536717761)

[**9.3 Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution** 34](#_Toc536717762)

[**9.4 Use java.util.concurrent collections and classes including CyclicBarrier and CopyOnWriteArrayList** 35](#_Toc536717763)

[**10. Localization** 36](#_Toc536717764)

[**10.1 Read and set the locale by using the Locale object** 36](#_Toc536717765)

[**10.2 Create and read a Properties file** 37](#_Toc536717766)

[**10.3 Build a resource bundle for each locale and load a resource bundle in an application** 38](#_Toc536717767)

[**11. Building Database Applications with JDBC** 39](#_Toc536717768)

[**11. 1 Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations** 39](#_Toc536717769)

[Connection: 39](#_Toc536717770)

[Statements: 39](#_Toc536717771)

[ResultSet: 39](#_Toc536717772)

[**11.2 Identify the components required to connect to a database using the DriverManager class including the JDBC URL** 39](#_Toc536717773)

[**11.3 Submit queries and read results from the database including creating statements, returning result sets, iterating through the results, and properly closing result sets, statements, and connections** 39](#_Toc536717774)

# **1. Java Class Design**

## **1.1 Implement encapsulation**

|  |
| --- |
| *public class Encapsulation {* |
| *static int employee\_id = 12268;* |
| *private int id;* |
| *//constructor overloading* |
| *Encapsulation (){* |
| *System.out.println("From default constructor");* |
| *}* |
| *Encapsulation (String name){* |
| *System.out.println("From parameterized constructor");* |
| *}* |
| *public int getId() {* |
| *return id;* |
| *}* |
| *public void setId(int id) {* |
| *this.id = id;* |
| *}* |
|  |
| *//Overloaded Method* |
| *public String getEmployeeCode(int recordId){* |
| *return "from employee code with one orgs";* |
| *}* |
|  |
| *public String getEmployeeCode(int recordId, String name){* |
| *return "from employee code with two orgs";* |
| *}* |
|  |
| *//static method* |
| *public static String getEmployeeCode(){* |
| *return "From static method";* |
| *}* |
| *}* |
| *public class EncapsulationImpl {* |
| *public static void main(String[] args) {* |
| *Encapsulation encap = new Encapsulation();* |
| *encap.setId(12172);* |
| *System.out.println("Encap :"+ encap.getId());* |
| *System.out.println(encap.getEmployeeCode(12181));* |
| *System.out.println(encap.getEmployeeCode(13014, "Uday"));* |
| *System.out.println(Encapsulation.getEmployeeCode());* |
| *Encapsulation encap\_p = new Encapsulation("Pramati");* |
| *}* |
| *}* |

## **1.2 Implement inheritance including visibility modifiers and composition**

Refer 1.3

## **1.3 Implement polymorphism**

|  |
| --- |
| *abstract class Pet {* |
| *public abstract void makeSound();* |
| *private void getPetsInfo(){* |
| *System.out.println("from private mathod");* |
| *}* |
| *}* |
|  |
| *public class Dog extends Pet {* |
| *@Override* |
| *public void makeSound() {* |
| *System.out.println("Woof");* |
| *}* |
| *}* |
|  |
| *public class Cat extends Pet{* |
| *@Override* |
| *public void makeSound() {* |
| *System.out.println("Meow");* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.ArrayList;* |
| *import java.util.List;* |
| *public class PolymorphismExample {* |
| *public static void main(String[] args) {* |
| *List<Pet> pets = new ArrayList<Pet>();* |
| *pets.add(new Dog());* |
| *pets.add(new Cat());* |
| *for (Pet pet : pets) {* |
| *pet.makeSound();* |
| *//pet.getPetsInfo(); //As it's private type Pet is not visible* |
| *}* |
| *}* |
| *}* |

## **1.4 Override hashCode, equals, and toString methods from Object class**

|  |
| --- |
| import java.util.Objects; |
| public class EqualsAndHashCode { |
| public String name; |
| public int 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; |
| } |
|  |
| @Override |
| public boolean equals(Object obj) { |
| if(!(obj instanceof EqualsAndHashCode)){ |
|  |
|  |
| return false; |
| } |
| EqualsAndHashCode info =(EqualsAndHashCode)obj; |
| return Objects.equals(name,info.name) && Objects.equals(id, info.id); |
| } |
|  |
| @Override |
| public int hashCode() { |
| return this.id; |
| } |
|  |
|  |
| @Override |
| public String toString() { |
| return name; |
| } |
| public static void main(String[] args) { |
| EqualsAndHashCode equalshash = new EqualsAndHashCode(); |
| equalshash.setName("Venki"); |
| equalshash.setId(12268); |
| EqualsAndHashCode equalshash\_impl = new EqualsAndHashCode(); |
| equalshash\_impl.setName("Venki"); |
| equalshash\_impl.setId(122689); |
|  |
| if (equalshash.hashCode() == equalshash\_impl.hashCode()) { |
| if (equalshash.equals(equalshash\_impl)) { |
| System.out.println("Both objects are equal"); |
| } |
| else System.out.println("Both Objects are not equal."); |
| } |
| else |
| System.out.println("Both Objects are not equal. "); |
|  |
| } |
| } |

## **1.5 Create and use singleton classes and immutable classes**

|  |
| --- |
| *import java.util.HashSet;* |
| *import java.util.Set;* |
| *public class SingletonDesignPattern {* |
| *//public static final SingletonDesignPattern INSTANCE = new SingletonDesignPattern();* |
| *private static SingletonDesignPattern INSTANCE;* |
| *private Set<String> availableSeats;* |
|  |
| *public static SingletonDesignPattern getInstance(){* |
| *if(INSTANCE == null){* |
| *INSTANCE = new SingletonDesignPattern();* |
| *}* |
| *return INSTANCE;* |
| *}* |
|  |
| *public SingletonDesignPattern() {* |
| *availableSeats = new HashSet<String>();* |
| *availableSeats.add("A1");* |
| *availableSeats.add("A2");* |
| *}* |
|  |
| *public boolean bookSeats(String seat){* |
| *System.out.println(availableSeats);* |
| *return availableSeats.remove(seat);* |
| *}* |
|  |
| *public static void ticketAgentBooks(String seat){* |
| *//SingletonDesignPattern pattern = new SingletonDesignPattern(); //with this every time it will create a new object* |
| *SingletonDesignPattern pattern = getInstance();* |
| *System.out.println(pattern.bookSeats(seat));* |
| *}* |
|  |
| *public static void main(String[] args) {* |
| *ticketAgentBooks("A1");* |
| *ticketAgentBooks("A2");* |
| *}* |
| *}* |
|  |
| ***Immutable Example:*** |
| *public final class ImmutableExample {* |
| *final String name;* |
| *final int empId;* |
| *public ImmutableExample(String name, int empid){* |
| *this.name=name;* |
| *this.empId=empid;* |
| *}* |
| *public String getname(){* |
| *return name;* |
| *}* |
| *public int getEmpID(){* |
| *return empId;* |
| *}* |
| *}* |
|  |
| *public class ImmutableTest{* |
| *public static void main(String[] args) {* |
| *ImmutableExample immutable = new ImmutableExample("Venki", 12268);* |
| *System.out.println(immutable.getEmpID());* |
| *System.out.println(immutable.getname());* |
| *}* |
| *}* |

## **1.5 Develop code that uses static keyword on initialize blocks, variables, methods, and classes**

|  |
| --- |
| *public class StaticKeyWordExample {* |
| *static int lab1 = 20;* |
| *static int lab2 =10;* |
| *static {* |
| *System.out.println("from static block");* |
| *System.out.println("lab1+lab2 :"+ (lab1+lab2));* |
| *}* |
|  |
| *public static void staticMethod() {* |
| *System.out.println("from static method");* |
| *}* |
|  |
| *public static void main(String[] args) {* |
| *staticMethod();* |
| *}* |
| *}* |

# **2. Generics and Collections**

## **2.1 Create and use a generic class**

|  |
| --- |
| public class GenericClassExample<T> { |
| private T t; |
| public T getT() { |
| return t; |
| } |
|  |
| public void setT(T t) { |
| this.t = t; |
| } |
| public static void main(String[] args) { |
| GenericClassExample<String> generic = new GenericClassExample<String>(); |
| generic.setT("Pramati"); |
| GenericClassExample generic1 = new GenericClassExample(); |
| generic1.setT("Pramati1"); |
| generic1.setT(20); |
| } |
| } |

## **2.2 Create and use ArrayList, TreeSet, TreeMap, and ArrayDeque objects**

### ArrayList:

ArrayList is an implementation of the List interface that internally uses an Array to store the elements.

### TreeSet:

TreeSet is an implementation of the Set interface that uses tree for storage. The elements are orders using their natural order.

### TreeMap:

TreeMap is an implementation of the Map interface that uses tree for storage. The elements are ordered using the natural ordering or their keys.

### ArrayDeque:

ArrayDeque is an implementation of the Deque interface. Array deques have no capacity restrictions; they grow as necessary to support usage. They are not thread-safe. Null elements are prohibited

|  |
| --- |
| *import java.util.ArrayDeque;* |
| *import java.util.ArrayList;* |
| *import java.util.Iterator;* |
| *import java.util.List;* |
| *import java.util.Map;* |
| *import java.util.Map.Entry;* |
| *import java.util.Set;* |
| *import java.util.TreeMap;* |
| *import java.util.TreeSet;* |
|  |
| *public class ArrayListExample {* |
| *public static void main(String[] args) {* |
| *List<String> list = new ArrayList <>();* |
| *list.add("Corridor");* |
| *list.add("WM");* |
| *list.add("SpotCues");* |
| *list.add("RaveMarketing");* |
| *list.add("LSUC");* |
| *list.add("Corridor");* |
| *System.out.println("List of Projects :"+ list);* |
|  |
| *//TreeSet* |
| *Set<String> treeSet = new TreeSet<>();* |
| *treeSet.add("Corridor");* |
| *treeSet.add("WM");* |
| *treeSet.add("SpotCues");* |
| *treeSet.add("RaveMarketing");* |
| *treeSet.add("LSUC");* |
| *treeSet.add("Corridor");* |
| *System.out.println("Projects :"+ treeSet);* |
|  |
| *//TreeMap* |
| *Map<Integer, String> tmap = new TreeMap<>();* |
| *tmap.put(12268, "Venkatesh");* |
| *tmap.put(12035, "Krishna");* |
| *tmap.put(12172, "Rana");* |
| *System.out.println("Employee Info : "+ tmap);* |
| *System.out.println("Employee Name by ID : "+ tmap.get(12268));* |
|  |
| *Set<Entry<Integer, String>> set = tmap.entrySet();* |
| *Iterator<Entry<Integer, String>> iterator = set.iterator();* |
| *while(iterator.hasNext()){* |
| *Entry<Integer, String> entry = iterator.next();* |
| *System.out.print("Key :"+entry.getKey());* |
| *System.out.println(" Value :"+entry.getValue());* |
| *}* |
|  |
| *//ArrayDeque* |
| *ArrayDeque<String> deque = new ArrayDeque<>();* |
| *deque.add("Corridor");* |
| *deque.add("WM");* |
| *deque.add("SpotCues");* |
| *deque.add("Rav Marketing");* |
| *deque.add("LSUC");* |
| *deque.add("Corridor");* |
| *System.out.println("From ArrayDeque : "+deque);* |
| *System.out.println("Getting 1st element : "+deque.element());* |
| *System.out.println(deque.poll());* |
| *System.out.println("From ArrayDeque : "+deque);* |
| *System.out.println(deque.removeFirst());* |
| *System.out.println("From ArrayDeque : "+deque);* |
| *System.out.println();* |
| *}* |
| *}* |

## **2.3 Use java.util.Comparator and java.lang.Comparable interfaces**

### Comparator:

|  |
| --- |
| *public class EmployeeInfo {* |
| *private String name;* |
| *private int empid;* |
| *private String email;* |
| *public String getName() {* |
| *return name;* |
| *}* |
| *public void setName(String name) {* |
| *this.name = name;* |
| *}* |
| *public int getEmpid() {* |
| *return empid;* |
| *}* |
| *public void setEmpid(int empid) {* |
| *this.empid = empid;* |
| *}* |
| *public String getEmail() {* |
| *return email;* |
| *}* |
| *public void setEmail(String email) {* |
| *this.email = email;* |
| *}* |
|  |
| *EmployeeInfo(String name, int id, String email){* |
| *this.email=email;* |
| *this.empid=id;* |
| *this.name=name;* |
| *}* |
| *@Override* |
| *public String toString() {* |
| *return "[id="+this.empid +"name="+this.name+"email="+this.email+"]";* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.Comparator;* |
| *public class EmpIdComparator implements Comparator<EmployeeInfo>{* |
| *@Override* |
| *public int compare(EmployeeInfo emp1, EmployeeInfo emp2) {* |
| *return emp1.getEmpid() - emp2.getEmpid();* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.Comparator;* |
| *public class NameComparator implements Comparator<EmployeeInfo>{* |
| *@Override* |
| *public int compare(EmployeeInfo emp1, EmployeeInfo emp2) {* |
| *String name = emp1.getName();* |
| *String nam2 = emp2.getName();* |
| *return name.compareTo(nam2);* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.ArrayList;* |
| *import java.util.Collections;* |
| *public class MainComparator {* |
| *public static void main(String[] args) {* |
| *ArrayList<EmployeeInfo> emp = new ArrayList<EmployeeInfo>();* |
| *emp.add(new EmployeeInfo("venki", 12268, "test@gmail.com"));* |
| *emp.add(new EmployeeInfo("Krishna", 13035, "test1@gmail.com"));* |
| *emp.add(new EmployeeInfo("Rana", 12172, "test2@gmail.com"));* |
|  |
| *System.out.println("Before order : ");* |
| *for (EmployeeInfo empinfo : emp) {* |
| *System.out.println(empinfo);* |
| *}* |
|  |
|  |
| *Collections.sort(emp, new NameComparator());* |
| *System.out.println("After NameComparator : ");* |
| *for (EmployeeInfo empinfo : emp) {* |
| *System.out.println(empinfo);* |
| *}* |
|  |
| *Collections.sort(emp, new EmpIdComparator());* |
| *System.out.println("After EmpIdComparator : ");* |
| *for (EmployeeInfo empinfo : emp) {* |
| *System.out.println(empinfo);* |
| *}* |
| *}* |
| *}* |

### Comparable:

|  |
| --- |
| *import java.util.ArrayList;* |
| *import java.util.Collections;* |
| *public class ComparableExample implements Comparable<ComparableExample>{* |
| *private String name;* |
| *private int empid;* |
| *private String email;* |
| *public String getName() {* |
| *return name;* |
| *}* |
| *public void setName(String name) {* |
| *this.name = name;* |
| *}* |
| *public int getEmpid() {* |
| *return empid;* |
| *}* |
| *public void setEmpid(int empid) {* |
| *this.empid = empid;* |
| *}* |
| *public String getEmail() {* |
| *return email;* |
| *}* |
| *public void setEmail(String email) {* |
| *this.email = email;* |
| *}* |
|  |
| *ComparableExample(String name, int id, String email){* |
| *this.email=email;* |
| *this.empid=id;* |
| *this.name=name;* |
| *}* |
|  |
| *@Override* |
| *public String toString() {* |
|  |
| *return "[id="+this.empid +"name="+this.name+"email="+this.email+"]";* |
| *}* |
|  |
|  |
| *public static void main(String[] args) {* |
| *ArrayList<ComparableExample> list = new ArrayList<ComparableExample>();* |
| *list.add(new ComparableExample("venki", 12268, "test@gmail.com"));* |
| *list.add(new ComparableExample("Krishna", 13035, "test1@gmail.com"));* |
| *list.add(new ComparableExample("Rana", 12172, "test2@gmail.com"));* |
|  |
| *Collections.sort(list);* |
| *System.out.println("After Sort");* |
| *for (ComparableExample empinfo : list) {* |
| *System.out.println(empinfo);* |
| *}* |
| *}* |
| *//used to sort details by name* |
| *@Override* |
| *public int compareTo(ComparableExample emp) {* |
|  |
| *return (this.name.compareTo(emp.name));* |
| *//return this.name. - emp.name;* |
| *}* |
| *}* |

# **4. Java Stream API**

## **4.1 Create and manage date-based and time-based events including a combination of date and time into a single object using LocalDate, LocalTime, LocalDateTime, Instant, Period, and Duratio**n

|  |
| --- |
| *import java.time.Duration;* |
| *import java.time.Instant;* |
| *import java.time.LocalDate;* |
| *import java.time.LocalDateTime;* |
| *import java.time.LocalTime;* |
| *import java.time.Month;* |
| *import java.time.Period;* |
|  |
| *public class DateAndTimeExample {* |
| *public static void main(String[] args) {* |
|  |
| *LocalDate date = LocalDate.now();* |
| *LocalTime time = LocalTime.now();* |
| *int day = LocalDate.now().getDayOfYear();* |
| *Month locMonth = LocalDate.now().getMonth();* |
|  |
| *LocalDateTime dateTime = LocalDateTime.now();* |
| *Month month = dateTime.getMonth();* |
| *System.out.println(dateTime);* |
| *System.out.println(month);* |
| *System.out.println(date);* |
| *System.out.println(day);* |
| *System.out.println(locMonth);* |
|  |
| *System.out.println(LocalDate.of(2019, 01, 03));* |
| *System.out.println(LocalTime.of(14,30)+" PM");* |
| *System.out.println(LocalDateTime.of(date, time));* |
| *System.out.println(time);* |
|  |
| *Duration duraion = Duration.ofSeconds(30, 20);* |
| *Duration duraion2 = Duration.between(LocalTime.NOON, LocalTime.MIDNIGHT);* |
| *System.out.println(duraion);* |
| *System.out.println(duraion2);* |
|  |
| *Period period = Period.of(2019, 01, 05);* |
| *Period periodOfMonths = Period.ofMonths(2);* |
| *Period period1 = Period.between(LocalDate.of(2019, 01, 05), LocalDate.of(2018, 10, 20));* |
| *System.out.println(period);* |
| *System.out.println(periodOfMonths);* |
| *System.out.println(period1);* |
|  |
| *Instant instant = Instant.now();* |
| *long instant1 = instant.getEpochSecond();* |
| *System.out.println(instant);* |
| *System.out.println(instant1);* |
| *}* |
| *}* |

## **4.2 Work with dates and times across time zones and manage changes resulting from daylight savings including Format date and times values**

Refer 4.1

## **4.3 Define and create and manage date-based and time-based events using Instant, Period, Duration, and TemporalUnit**

Refer 4.3

# **5. Java File I/O (NIO.2)**

## **5.1 Use Path interface to operate on file and directory paths**

Refer 5.2

## **5.2 Use Files class to check, read, delete, copy, move and manage metadata of a file or directory**

|  |
| --- |
| *import java.io.IOException;* |
| *import java.nio.file.Files;* |
| *import java.nio.file.Path;* |
| *import java.nio.file.Paths;* |
| *public class PathExample {* |
| *public static void main(String[] args) throws IOException {* |
| *Path path = Paths.get("tracker.txt");* |
| *System.out.println(Files.exists(path));* |
| *if(!Files.exists(path)){* |
| *Files.createFile(path);* |
| *}* |
| *System.out.println(Files.exists(path));* |
| *Path source = Paths.get("E:/Practice/JavaBasics/temp1.txt");* |
| *Path target = Paths.get("E:/Practice/JavaBasics/temp2.txt");* |
|  |
| *if(!Files.exists(source)){* |
| *Files.createFile(source);* |
| *}* |
| *if(!Files.exists(target)){* |
| *Files.createFile(target);* |
| *}* |
| *//Files.copy(source, target);* |
| *//Files.move(source, target);* |
| *Files.delete(path);* |
| *Files.deleteIfExists(path);* |
| *}* |
| *}* |

# **6. Advanced Java Class Design**

## **6.1 Develop code that uses abstract classes and methods**

|  |
| --- |
| *public abstract class AbstractCalssExample{* |
| *abstract void getName();* |
| *abstract void getCount();* |
| *public void getBusinessUnit(){* |
| *System.out.println("Imaginea");* |
| *}* |
| *}* |

|  |
| --- |
| public interface Interfacexample { |
| public static void getStaticMessage(){ |
| System.out.println("from interface static"); |
| } |
| public void getMessage(); |
| } |
|  |
| public class AbstarctImpl extends AbstractCalssExample implements Interfacexample{ |
| @Override |
| void getName() { |
| System.out.println("Pramati"); |
| } |
|  |
| @Override |
| void getCount() { |
| System.out.println("count"); |
| } |
| @Override |
| public void getMessage() { |
| System.out.println("get message"); |
| } |
| } |

|  |
| --- |
| *public class AbstractAndInterfaceExample {* |
| *public static void main(String[] args) {* |
| *AbstractCalssExample impl = new AbstarctImpl();* |
| *impl.getBusinessUnit();* |
| *impl.getCount();* |
| *((AbstarctImpl) impl).getMessage();* |
| *impl.getName();* |
| *}* |
| *}* |

## **6.2 Develop code that uses the final keyword**

|  |
| --- |
| *public class FinalKeyWordExample {* |
| *static final int count=100;* |
| *void getCount(){* |
| *System.out.println("count: "+count);* |
| *}* |
| *public static void main(String[] args) {* |
| *System.out.println(FinalKeyWordExample.count);* |
| *new FinalKeyWordExample().getCount();* |
| *}* |
| *}* |

**Example 2:**

|  |
| --- |
| *import java.util.HashSet;* |
| *import java.util.Set;* |
| *public class SingletonDesignPattern {* |
|  |
| *public static final SingletonDesignPattern INSTANCE = new SingletonDesignPattern();* |
| *//private static SingletonDesignPattern INSTANCE;* |
| *private Set<String> availableSeats;* |
|  |
| */\*public static SingletonDesignPattern getInstance(){* |
| *if(INSTANCE == null){* |
| *INSTANCE = new SingletonDesignPattern();* |
| *}* |
| *return INSTANCE;* |
| *}\*/* |
|  |
| *public SingletonDesignPattern() {* |
| *availableSeats = new HashSet<String>();* |
| *availableSeats.add("A1");* |
| *availableSeats.add("A2");* |
| *}* |
|  |
| *public boolean bookSeats(String seat){* |
| *System.out.println(availableSeats);* |
| *return availableSeats.remove(seat);* |
|  |
| *}* |
|  |
| *public static void ticketAgentBooks(String seat){* |
| *SingletonDesignPattern pattern = new SingletonDesignPattern(); //with this every time it will create a new object* |
| *//SingletonDesignPattern pattern = getInstance();* |
| *System.out.println(pattern.bookSeats(seat));* |
|  |
| *}* |
| *public static void main(String[] args) {* |
| *ticketAgentBooks("A1");* |
| *ticketAgentBooks("A2");* |
|  |
| *}* |
| *}* |

## **6.3 Create inner classes including static inner class, local class, nested class, and anonymous inner class**

|  |
| --- |
| *interface Eatable{* |
| *void eat();* |
| *}* |
|  |
| *public class InnerCalssesExample {* |
| *static int count = 20;* |
| *int val =10;* |
| *//static inner class* |
| *static class InnerClass{* |
| *void getMsg(){* |
| *System.out.println("count from static inner class: "+count);* |
| *}* |
| *}* |
|  |
| *void getMsg(){* |
| *//Local Inner class* |
| *class LocalInnerClass{* |
| *void getMsg(){* |
| *System.out.println("value from static inner class: "+val);* |
| *}* |
|  |
| *}* |
| *LocalInnerClass innerClass = new LocalInnerClass();* |
| *innerClass.getMsg();* |
| *}* |
| *public static void main(String[] args) {* |
| *InnerCalssesExample.InnerClass staticInner = new InnerCalssesExample.InnerClass();* |
| *staticInner.getMsg();* |
| *new InnerCalssesExample().getMsg();* |
|  |
| *//anonymous inner class* |
| *Eatable e=new Eatable(){* |
| *public void eat()* |
| *{* |
| *System.out.println("nice fruits");* |
| *}* |
| *};* |
| *e.eat();* |
| *}* |
| *}* |

## **6.4 Use enumerated types including methods, and constructors in an enum type**

|  |
| --- |
| *enum ResumeStatus {* |
| *SUCESS, IN\_PROGRESS, NEW* |
| *}* |
| *public class EnumExample {* |
|  |
| *public static void main(String[] args) {* |
| *ResumeStatus status = ResumeStatus.IN\_PROGRESS;* |
| *System.out.println(status);* |
| *}* |
| *}* |

## **6.5 Develop code that declares, implements and/or extends interfaces and use the @Override annotation**

|  |
| --- |
| *public interface Interfacexample {* |
| *public static void getStaticMessage(){* |
| *System.out.println("from interface static");* |
| *}* |
| *public void getMessage();* |
| *}* |
| *public class AbstarctImpl extends AbstractCalssExample implements Interfacexample{* |
| *@Override* |
| *void getName() {* |
| *System.out.println("Pramati");* |
| *}* |
| *@Override* |
| *void getCount() {* |
| *System.out.println("count");* |
| *}* |
| *@Override* |
| *public void getMessage() {* |
| *System.out.println("get message");* |
| *}* |
| *}* |

|  |
| --- |
| *public abstract class AbstractCalssExample{* |
| *abstract void getName();* |
| *abstract void getCount();* |
| *public void getBusinessUnit(){* |
| *System.out.println("Imaginea");* |
| *}* |
| *}* |
| *public class AbstractAndInterfaceExample {* |
| *public static void main(String[] args) {* |
| *AbstractCalssExample impl = new AbstarctImpl();* |
| *impl.getBusinessUnit();* |
| *impl.getCount();* |
| *((AbstarctImpl) impl).getMessage();* |
| *impl.getName();* |
| *}* |
| *}* |

# **7. Exceptions and Assertions**

## **7.1 Use try-catch and throw statements**

|  |
| --- |
| *public class TryCatchExample {* |
| *public static void main(String[] args) {* |
| *int val1 =0;* |
| *int val2 =20;* |
| *try {* |
| *int val3= val2/val1;* |
| *System.out.println("from try block");* |
|  |
| *} catch (ArithmeticException e) {* |
| *System.out.println("Divide by zero");* |
| *}* |
| *}* |
|  |
| *}* |

## **7.2 Use catch, multi-catch, and finally clauses**

|  |
| --- |
| import java.io.BufferedReader; |
| import java.io.File; |
| import java.io.FileReader; |
| import java.io.FileWriter; |
| import java.io.IOException; |
| import java.io.PrintWriter; |
|  |
| public class FileExample { |
| public static void main(String[] args) throws IOException { |
| FileReader reader = null; |
| try { |
| File mkdir = new File("makdir"); |
| mkdir.mkdir(); |
| File newfile = new File(mkdir, "sample\_file.txt"); |
| newfile.createNewFile(); |
| PrintWriter writer = new PrintWriter(newfile); |
| writer.println("sample code"); |
| writer.flush(); |
| writer.close(); |
| reader = new FileReader(newfile); |
| BufferedReader br = new BufferedReader(reader); |
| System.out.println(br.readLine()); |
| mkdir.delete(); |
| } catch (IOException e) { |
| e.printStackTrace(); |
| } |
| finally { |
| reader.close(); |
| } |
| } |
| } |

## **7.3 Use Autoclose resources with a try-with-resources statement**

Refer 8.2

## **7.4 Create custom exceptions and Auto-closeable resources**

|  |
| --- |
| *public class IncorrectFileNameException extends Exception {* |
| *public IncorrectFileNameException(String errorMessage, Throwable error){* |
| *super(errorMessage, error);* |
| *}* |
| *}* |

|  |
| --- |
| import java.io.File; |
| import java.io.FileNotFoundException; |
| import java.util.Scanner; |
|  |
| public class CustomExceptionExample { |
| public static void main(String[] args) throws IncorrectFileNameException { |
| File file = null; |
| try { |
| file= new File("E:/Practice/JavaBasics/temp1"); |
| Scanner scanner = new Scanner(file); |
| if(scanner.hasNextLine()){ |
| System.out.println(scanner.nextLine()); |
|  |
| } |
|  |
| } catch (FileNotFoundException error) { |
|  |
| throw new IncorrectFileNameException("File exception :"+file, error); |
| } |
| } |
| } |

## **7.5 Test invariants by using assertions**

|  |
| --- |
| *public class AssertionExample {* |
| *public static void main(String[] args) {* |
| *int count = 20;* |
| *assert count > 50 : "Count exceeded";* |
| *System.out.println("Count : "+count);* |
| *}* |
| *}* |

# **8. Java I/O Fundamentals**

## **8.1 Read and write data from the console**

|  |
| --- |
| *import java.io.Console;* |
| *public class ConsoleExample {* |
|  |
| *public static void main(String[] args) {* |
| *Console console =System.console();* |
| *String userName = console.readLine("Please enter Username :");* |
|  |
| *char[] pw = console.readPassword("%S", "PWD:");* |
| *System.out.println(userName);* |
| *}* |
| *}* |

## **8.2 Use BufferedReader, BufferedWriter, File, FileReader, FileWriter, FileInputStream, FileOutputStream, ObjectOutputStream, ObjectInputStream, and PrintWriter in the java.io package.**

|  |
| --- |
| *import java.io.BufferedReader;* |
| *import java.io.BufferedWriter;* |
| *import java.io.File;* |
| *import java.io.FileReader;* |
| *import java.io.FileWriter;* |
| *import java.io.IOException;* |
| *import java.io.PrintWriter;* |
| *import java.io.Writer;* |
|  |
| *public class FileReaderExample {* |
| *public static void main(String[] args) throws IOException {* |
|  |
| *char[] in =new char[20];* |
| *File file = new File("file.txt");* |
| *boolean fileExists=file.exists();* |
| *boolean fineRenamed = file.renameTo(new File("newfile.txt"));* |
|  |
| *FileWriter writer = new FileWriter(file);* |
| *writer.write("Pramati Technologies");* |
| *writer.close();* |
|  |
| *FileReader reader = new FileReader(file);* |
| *reader.read(in);* |
| *for (char c : in) {* |
| *System.out.print(c);* |
| *}* |
|  |
| *BufferedReader bufferreader = new BufferedReader(new FileReader(file));* |
| *String data = bufferreader.readLine();* |
| *System.out.println();* |
| *System.out.println("From BufferReader: "+data);* |
|  |
|  |
| *File file2 = new File("bufferwriter.txt");* |
| *BufferedWriter brwritter = new BufferedWriter(new FileWriter(file2));* |
| *brwritter.write("Pramati Technologies");* |
|  |
| *PrintWriter pr = new PrintWriter(new FileWriter(file2));* |
| *pr.println("test");* |
| *pr.println("test2");* |
| *}* |
| *}* |

# **9. Java Concurrency**

## **9.1 Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks**

|  |
| --- |
| *public class ThreadExample implements Runnable{* |
| *@Override* |
| *public void run() {* |
| *System.out.println("from run");* |
| *System.out.println("from :" + Thread.currentThread().getName());* |
| *for (int i = 0; i < 4; i++) {* |
| *System.out.println("from :" + Thread.currentThread().getName()+ " I ="+i);* |
| *}* |
| *}* |
| *}* |

|  |
| --- |
| *public class TestThread {* |
| *public static void main(String[] args) {* |
| *ThreadExample thread = new ThreadExample();* |
| *Thread t = new Thread(thread);* |
| *Thread t1 = new Thread(thread);* |
| *Thread t2 = new Thread(thread);* |
| *t.setName("pramati");* |
| *t1.setName("WM");* |
| *t2.setName("Spotcues");* |
| *t.start();* |
| *t1.start();* |
| *t2.start();* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.concurrent.Callable;* |
| *public class CallableExample implements Callable<Integer>{* |
| *@Override* |
| *public Integer call() throws Exception {* |
| *int count = 100;* |
| *return count;* |
| *}* |
| *}* |

|  |
| --- |
| *import java.util.concurrent.Callable;* |
| *import java.util.concurrent.ExecutionException;* |
| *import java.util.concurrent.ExecutorService;* |
| *import java.util.concurrent.Executors;* |
| *import java.util.concurrent.Future;* |
|  |
| *public class TestCallable {* |
| *public static void main(String[] args) {* |
| *ExecutorService executorService =null;* |
| *try {* |
| *Callable callable = new CallableExample();* |
| *//Executor* |
| *executorService = Executors.newSingleThreadExecutor();* |
| *Future future = executorService.submit(callable);* |
| *System.out.println("from call method : "+future.get());* |
| *} catch (InterruptedException | ExecutionException e) {* |
| *e.printStackTrace();* |
| *}* |
| *executorService.shutdown();* |
| *}* |
| *}* |

## **9.2 Identify potential threading problems among deadlock, starvation, livelock, and race conditions**

### Deadlock:

Deadlock is the situation where two or more threads are blocked forever and waiting for each other.

### Starvation Lock:

Starvation occurs when a thread having least priority compare to other ones.

### Livelock:

Livelock is like deadlock, means that two are more threads are blocking each other. But with livelock each thread waiting actively to resolve the problem on its own.

### Race condition:

Race condition is the situation where two threads compete for the same value/resource and they try to change it on the same time.

## **9.3 Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution**

|  |
| --- |
| *import java.util.concurrent.atomic.AtomicInteger;* |
|  |
| *public class SynchronizedExample {* |
| *int count =20;* |
| *void getCount(){* |
| *synchronized (this) {* |
| *this.count = this.count+1;* |
| *}* |
| *}* |
| *public static void main(String[] args) {* |
| *AtomicInteger ai = new AtomicInteger(20);* |
| *int val = ai.get();* |
| *System.out.println(val);* |
| *int expectedval = 20;* |
| *int newVal = 10;* |
| *System.out.println(ai.compareAndSet(expectedval, newVal));* |
| *System.out.println(ai);* |
| *val = ai.getAndAdd(10);* |
| *System.out.println("ai :"+ai+" val : "+val);* |
| *}* |
| *}* |

## **9.4 Use java.util.concurrent collections and classes including CyclicBarrier and CopyOnWriteArrayList**

|  |
| --- |
| *import java.util.Iterator;* |
| *import java.util.concurrent.BrokenBarrierException;* |
| *import java.util.concurrent.CopyOnWriteArrayList;* |
| *import java.util.concurrent.CyclicBarrier;* |
| *import java.util.concurrent.TimeUnit;* |
| *import java.util.concurrent.TimeoutException;* |
|  |
| *public class ConcurrentExample {* |
|  |
| *public static void main(String[] args) throws InterruptedException, BrokenBarrierException, TimeoutException {* |
|  |
| *//Anonymous Inner type* |
| *Runnable runnable = new Runnable() {* |
|  |
| *@Override* |
| *public void run() {* |
| *System.out.println("From Anonymous run method");* |
|  |
| *}* |
| *};* |
| *CopyOnWriteArrayList list = new CopyOnWriteArrayList<>();* |
| *list.add("Venki");* |
| *list.add(12268);* |
| *list.add("test@gmail.com");* |
| *Iterator iterator = list.iterator();* |
| *while (iterator.hasNext()){* |
| *System.out.println(iterator.next());* |
| *}* |
| *CyclicBarrier cbarrier = new CyclicBarrier(2);* |
| *//cbarrier.await();* |
| *cbarrier.await(10, TimeUnit.SECONDS);* |
| *}* |
| *}* |

# **10. Localization**

## **10.1 Read and set the locale by using the Locale object**

|  |
| --- |
| *import java.util.Locale;* |
| *public class LocaleExample {* |
| *public static void main(String[] args) {* |
| *Locale locale = new Locale("en");* |
| *Locale locale1 = new Locale("en", "IN");* |
| *Locale locale2 = Locale.getDefault();* |
| *System.out.println(locale);* |
| *System.out.println(locale1);* |
| *System.out.println(locale2);* |
| *System.out.println(locale1.getLanguage());* |
| *System.out.println(locale2.getLanguage());* |
| *System.out.println(locale1.getCountry());* |
| *System.out.println(locale2.getCountry());* |
| *System.out.println(locale1.getDisplayName());* |
| *System.out.println(locale1.getISO3Language());* |
| *}* |
| *}* |

## **10.2 Create and read a Properties file**

|  |
| --- |
| *import java.io.FileInputStream;* |
| *import java.io.FileOutputStream;* |
| *import java.io.IOException;* |
| *import java.util.Properties;* |
|  |
| *public class CreateAndReadProperties {* |
| *public static void main(String[] args) {* |
| *Properties props = null;* |
| *FileOutputStream outStream =null;* |
| *FileInputStream inStream= null;* |
|  |
| *try {* |
| *outStream= new FileOutputStream("application.properties");* |
| *props= new Properties();* |
| *props.put("QB\_ID", "Test\_id");* |
| *props.put("token", "test\_token");* |
| *props.put("filed\_id", "206");* |
| *props.store(outStream, "Test properties");* |
|  |
| *inStream = new FileInputStream("application.properties");* |
| *props.load(inStream);* |
|  |
| *System.out.println(props.getProperty("QB\_ID"));* |
| *System.out.println(props.getProperty("token"));* |
| *System.out.println(props.getProperty("filed\_id"));* |
|  |
| *} catch (IOException e) {* |
|  |
| *e.printStackTrace();* |
| *}* |
| *finally {* |
| *try {* |
| *outStream.close();* |
| *inStream.close();* |
| *} catch (IOException e) {* |
| *e.printStackTrace();* |
| *}* |
|  |
| *}* |
| *}* |

## **10.3 Build a resource bundle for each locale and load a resource bundle in an application**

|  |
| --- |
| *import java.io.FileInputStream;* |
| *import java.io.FileNotFoundException;* |
| *import java.io.IOException;* |
| *import java.util.PropertyResourceBundle;* |
| *import java.util.ResourceBundle;* |
|  |
| *public class ResourceBundleExample {* |
| *public static void main(String[] args) {* |
| */\*Locale locale = new Locale("en", "IN");* |
| *ResourceBundle resource = ResourceBundle.getBundle("LabelsBundle", locale);\*/* |
| *FileInputStream fis;* |
| *try {* |
| *fis = new FileInputStream("LabelsBundle.properties");* |
| *ResourceBundle resource;* |
| *resource = new PropertyResourceBundle(fis);* |
| *System.out.println(resource.getString("s1"));* |
|  |
| *} catch (FileNotFoundException e) {* |
| *// TODO Auto-generated catch block* |
| *e.printStackTrace();* |
| *}* |
| *catch (IOException e) {* |
| *// TODO Auto-generated catch block* |
| *e.printStackTrace();* |
| *}* |
|  |
| *}* |
| *}* |

# **11. Building Database Applications with JDBC**

## **11. 1 Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations**

### Connection:

Connection is the session b/w application and Database.

### Statements:

Creates statement object to execute queries on the database.

### ResultSet:

It will process the data which returned from the Query we have executed.

## **11.2 Identify the components required to connect to a database using the DriverManager class including the JDBC URL**

Refer 12.3

## **11.3 Submit queries and read results from the database including creating statements, returning result sets, iterating through the results, and properly closing result sets, statements, and connections**

|  |
| --- |
| *import java.sql.Statement;* |
| *public class JDBCExample {* |
| *static final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";* |
| *static final String DB\_URL = "jdbc:mysql://localhost:3306/Employee";* |
|  |
| *static final String USERNAME = "root";* |
| *static final String PASSWORD = "root";* |
|  |
| *public static void main(String[] args) {* |
|  |
| *Connection conn = null;* |
| *Statement stmt = null;* |
|  |
| *try{* |
| *Class.forName(JDBC\_DRIVER);* |
|  |
| *System.out.println("Connecting to database...");* |
| *conn = DriverManager.getConnection(DB\_URL,USERNAME, PASSWORD);* |
| *stmt = conn.createStatement();* |
| */\** |
| *//To create database* |
| *System.out.println("Creating database");* |
| *String sql = "CREATE DATABASE IF NOT EXISTS EMPLOYEE";* |
| *stmt.executeUpdate(sql);* |
| *System.out.println("Database created successfully");\*/* |
|  |
| */\* //to crete table* |
| *String createTable = "CREATE TABLE IF NOT EXISTS USERS" +"(Empid INTEGER not NULL, " +* |
| *" Name VARCHAR(100), " +* |
| *" Email VARCHAR(100), " +* |
| *" Location VARCHAR(100), " +* |
| *" PRIMARY KEY ( Empid ))";* |
| *System.out.println("Creating table");* |
| *stmt.executeUpdate(createTable);* |
| *System.out.println("Table created");\*/* |
|  |
| *System.out.println("Inserting record into table");* |
| *String insertQuesry = "INSERT INTO USERS "+ "VALUES(12268,'Venkateswarlu','venkateswarlu.kakani@imaginea.com','Hyderabad')";* |
| *stmt.executeUpdate(insertQuesry);* |
| *System.out.println("Record inserted successfully");* |
| *String getEmployeeInfo = "SELECT \* FROM USERS WHERE EMPID=12268";* |
| *ResultSet result = stmt.executeQuery(getEmployeeInfo);* |
| *while(result.next()){* |
| *System.out.println(result.getString("email"));* |
| *}* |
|  |
| *stmt.close();* |
| *conn.close();* |
| *}* |
| *catch(SQLException se){* |
| *se.printStackTrace();* |
| *}catch(Exception e){* |
| *e.printStackTrace();* |
| *}* |
| *}* |
| *}* |