**FIVE DAY INTENSIVE JAVA - COURSE – DAY 1 - SUNIL KUMAR GUNASEKARAN – ISTQB CERTIFIED – November 15 2017 Friday 1746 HRS**

**To be Added -**

**Vignesh Inputs on Project structure and so on -**

**Three big projects in Java**

**Handling applets**

**JAVA SYLLABUS - DAY 1**

* Java Keywords
* Java, IDE, maven installation
* New, Class, Object
* Access modifiers
* Constructors
* Inheritance
* Polymorphism
* Abstract Class and Abstract Method
* Interfaces
* Garbage Collection
* Package and Import
* Stack Trace
* Exception Handling
* Assignment Problems

**1) Java Keywords**

|  |
| --- |
| In Java, a keyword is a word with a predefined meaning in Java programming language syntax. Reserved for Java, keywords may not be used as identifiers for naming variables, classes, methods or other entities.  boolean, byte, short, char, double, int, long, float -  Defines a character variable capable of holding any character of the java source file's character set. |

|  |
| --- |
|  |

**1) Java, IDE. Maven installation**

1. Install Java 7 SDK
2. Install Eclipse
3. Install Intellij Community Edition
4. Install TestNG Plugin for Eclipse
5. Install Git Bash
6. Git Maven Commands List
7. Install Maven for Windows
8. Import project in eclipse
9. Creating Git branch from Stash
10. Creating a pull request

**1) Install Java 7 SDK**

|  |
| --- |
| Java 8 is having compatibility issues with maven.  Install Java 7 from below URL.  <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>  a) Setting Path:  Set PATH in user variable section as “C:\Program Files\Java\jdk1.7.0\_67\bin”  b) Set “JAVA\_HOME” in system variable section  as C:\Program Files\Java\jdk1.7  c) Set following value for “path” in system variable section as “C:\Program Files\Java\jdk1.7.0\_67\bin”  d) Execute java -version to validate install. |

**2) Install Eclipse**

|  |
| --- |
| [**https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/lunar**](https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/lunar) |

**3) Install Intellij community edition**

|  |
| --- |
| Download intellij community edition from this url:  <http://www.jetbrains.com/idea/download/> |

**4) Install TestNG Plugin for eclipse**

|  |
| --- |
| TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:  Open eclipse then go to Help > Install New Software  Then type <http://beust.com/eclipse/> in the “work with” field and select TestNG and Install the plugin. |

**5) Install Git Bash**

|  |
| --- |
| [**http://git-scm.com/download/win**](http://git-scm.com/download/win) |

**6) Git / Maven Commands sheet:-**

|  |
| --- |
| 1. To know status: git status 2. To add files: git add -A 3. To commit: git commit -m “type message here” 4. To Push: git push  -u origin “branch name” 5. To Clone project: git clone  [ssh://git@devops-tools.pearson.com/occ/classroom-ui.git](mailto:ssh://git@devops-tools.pearson.com/occ/classroom-ui.git) 6. To check out another branch: git checkout “branch name here” 7. To pull changes from origin: git pull 8. Maven Execution: mvn clean install -Dmaven.test.skip=true 9. To register your name and email on stash:                  git config --global user.name “your name here"git                 config --global user.email "your email here" |

**7) Install Maven for Windows.**

|  |
| --- |
| a) Download Apache Maven  Visit this Maven [official website](http://maven.apache.org/download.html), choose a version and click on the download link, e.g apache-maven-2.2.1-bin.zip.  b) Extract It  Extract the downloaded zip file. In this case, we extracted to d driver and renamed the folder, e.g D:\maven.  c) Add MAVEN\_HOME  Add a new MAVEN\_HOME variable to the Windows environment, and point it to your Maven folder.  d) Add PATH  Update PATH variable, append “Maven bin folder” path, so that you can run the Maven’s command everywhere.  e) Verification  Done, to verify it, in command prompt, type “mvn –version“.  C:\Documents and Settings\mkyong>mvn -version Apache Maven 2.2.1 (r801777; 2009-08-07 03:16:01+0800) Java version: 1.6.0\_13 Java home: C:\Program Files\Java\jdk1.6.0\_13\jre Default locale: en\_US, platform encoding: Cp1252 OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"  If you see similar message, means your Apache Maven is installed successfully on Windows. |

**8)  Import project in Eclipse/Intellij:-**

|  |
| --- |
| a) Follow troubleshooting steps above to resolve importing issues.  b) Import projects as existing maven project.  c) Run a stable test suite and verify stability of test. End point - tests pass, browser loading as expected, no errors reported in logs.  d) Create a practice branch if you have rights. Otherwise use “branch-name/sunil-practice” to run, create new tests for execution.  e) To execute tests in intellij, select particular xml test suite in abd-tests\...\resources\testng-suites. Right click, select create xml config.  Config pops up for that particular xml suite. Select Suite radio button in configuration tab, in JDK settings, Append the following to VM Options, click apply and ok. In working directory field set the project folder. Select access module dependencies.  Use classpath of modules – select "abd" module.  -Dconfigfiles=test-users/educator4.xml,pi-staging-settings.xml,staging-settings.xml,web-test-settings.xml,base-settings.xml -Dbrowser=firefox  f) Double check above steps and hit triangle button as seen below. The configuration points to the xml selected whose configuration was done in the above steps.  g) If everything goes well, we see the following - testNg tests running in left, console logs in right, progress bar and test completion ratio. |

**9) Creating Git branch from Stash:-**

|  |
| --- |
| a) Click more button next to branch in the branch icon and click create branch from here button. The create branch repository, branch from and branch to should be selected and create branch button should be clicked. |

**10) Creating a pull request**

|  |
| --- |
| a) Go to project in stash  b) Click pull request button in the right hand column as seen below  c) In the pull request dashboard, select source branch – and destination branch. Add reviewers (Eg. ABD). Click :Create pull request” button    d) Once you open pull request, we can see request number, source and destination branch, options to merge and approve, modified files encased in their folders, actual code modifications and ability to comment on changes. If there is a merge conflict do manual merge using Intellij. |

**1) New operator, Class and Object:-**

|  |
| --- |
| New:-  The new operator dynamically allocates memory for an object during run time.  Advantages:-   * Your program can create as many or as few objects as it needs during the evaluation of your program.   Dis-Advantages:-   * Since memory is finite, it is possible that new will not be able to allocate memory for an object because insufficient memory exists. If this happens, a run-time exception will occur. |

|  |
| --- |
| Class:-  A class creates a new data type that can be used to create objects. That is, a class creates a logical framework that defines the relationship between its members.  A class is a blueprint containing members/data types and methods. |

|  |
| --- |
| Object:-  Object is an instance of that class. Thus, a class is a logical construct. An object has physical reality. (That is, an object occupies space in memory.) |

**2) Assembly Class Access Modifiers**

Access modifiers are those which **set access levels** for classes, attributes, methods and constructors.

|  |
| --- |
| **Assembly Class Java Access Modifiers for Class Level**   * Default = Accessed only within the package. * Public = Accessed outside the package. * Final = Final class level it cannot be sub classed but can be instantiated. * Abstract = Abstract has to be sub classed and not instantiated. |

|  |
| --- |
| **Assembly Class Access Modifiers for Attributes Level**   * Public = Attribute can be accessed by class/objects in any package * Private = Attribute can be accessed from only within the class. * Protected = Attributes can be accessed from within the same package and the sub classes outside the package. * Default = Attributes can be accessed by same class or other classes/objects within the same package. * Final = The value of the attribute cannot be changed. Only one value can be assigned * Static = Only one value of the attribute per class. Remains the same in all the objects |

|  |
| --- |
| Assembly Class Access Modifiers = Method Level   * Public = The method can be accessed by a class or object in any package * Private = Method can be accessed from only within the class * Protected = Method can be accessed by other classes or objects within the same package and the class/objects sub-classing this methods’ class outside the package * Default = Method can be accessed only be the other classes/objects within the same package and the class/objects sub-classing this methods’ class outside the package. * Final = Method cannot be overridden * Abstract = Only provides the method declaration not the definition. It can be inherited and can be overridden by sub classes. * Synchronized = Only one thread can access the method at a time   Static = They are those class level methods which don’t belong to any particular instance. In a static method there cannot be an instance variable; A static method is not allowed to read or write the non-static methods of its class. |

|  |
| --- |
| Assembly Class Final  Final Class = Cannot be extended within and outside the package  Final Function = Cannot be overridden in the sub class.  Final Variable = Value is like constant; New value cannot be assigned or modified anywhere. |

|  |
| --- |
| Class Diagram for a Class Attributes and Methods of Washing Machine and ER Diagram  Washing Machine color: make: size: model: spin() rinse() model() |

|  |
| --- |
| Explain “static” access modifier as applied to attributes, methods and classes.   * Static attribute – It is like a constant. Value of the variable cannot change inside the class or inside the package. Only one value can be assigned to it. And it remains constant throughout the class or package. * Static method – can be called directly without creation of object. E.g. public static void main. Can be called directly without requirement of object creation. * Static class – Static is applied to inner classes present inside an outer class. Only nested classes can be static. Outer class cannot be static. Nested classes no need reference of Outer class but inner classes do need reference of outer class. Nested classes can directly access static members but not non static members of outer class. |

|  |
| --- |
| **3) Constructors**   * A constructor initializes attributes after object is created. * It has same name as that of the class in which it resides, Once defined the constructor is automatically called immediately after the object creation. * No return type * Constructor’s job is to initialize the inner state of an object so that the code creating the instance of the class will have a fully initialized usable object immediately. * Two Types * Default Constructor – When you do not explicitly define a constructor for a class, then java creates a constructor for the class * Parameterized Constructor – When the object is created, the user can send the values to the constructor to initialize the attributes. |

**Example:-**

|  |
| --- |
| **public class Person {**  **int age;**  **short weight;**  **Person(int a, short w) {**  **age = a;**  **weight = w; } }**  **When the object is created, it is like>> Person Sunil = new Person(25,75);** |

**Copy Constructor**

* Frequently we need to use the same values used in the different object to initialize the current object. In that case we need to pass the object itself as the parameter.

|  |
| --- |
| **public CubePrime {**  **double w; double h; double l;**  **Cube(double a,double b, double c)**  **{ w=a; h=b; l=c; }**  **Cube (Cube obj) { w = obj.w; h= obj.h; l= obj.l; }**  **public static void main (String args) {**  **Cube obj1 = new Cube(10,20,30);**  **Cube obj2 = new Cube(obj1); } }** |

**Static Methods**

* Static methods are class level methods which don’t belong to one particular instance. Instance method and Instance variable versus Static method and static variable;

**4) Inheritance**

* A class has data and methods. Inheritance is how a class can get data and methods present in another class. Using extends keyword.

|  |
| --- |
| **public class A {**  **int i,j;**  **void showij() {**  **System.out.println(i+""+ j); } }**    **public class B extends A {**  **int k;**  **void showk() { System.out.println(k);}**  **void sum() { int sum = i+j+k; System.out.println(sum); } }**  **public static void main(String args[]) {**  **A superOb = new A();**  **B subOb = new B(); // Below can be done since B inherits the members and methods of A**  **B.i = 7; B.j = 8; B.k= 9;**  **B.sum();// Outputs 24 as result } }** |

Java does not support multiple Inheritance.

**Polymorphism**

|  |
| --- |
| **Overloading Methods is Compile Time Polymorphism**   * When we define two or more methods within the same class that share the same name but their parameter declaration are different the methods are said to be overloaded and the process is said to be method overloading. * In own words suppose there are three methods namely add(double a, double b) and add ()   And add (int a, int b). All three have same name but different parameters. When the object of the class is defined and called as obj.add() obj.add(10.2,10.4) or obj (1,3) the respective methods are called which have no parameters, double parameters and int parameters respectively |

**Overloading Constructors is also Polymorphism**

* Constructors can also be overloaded;

**Example**:-

|  |
| --- |
| **public class Cube {**  **double width; double height; double length;**  **// Three overloaded constructors**  **Cube() { width = -1; height =-1; length = -1; }**  **Cube (double x) {width = x; height = x; length = x;}**  **Cube(double w,double h,double l) {width = w; height=h; length = l;}**  **public static void main (String args[]) { // All three are valid**  **Cube c1 = new Cube();**  **Cube c2 = new Cube(3.2);**  **Cube c3= new Cube (3.1, 2.4, 4.5); } }** |

**Method Overriding**

|  |
| --- |
| * When the method with same name is present in the super class and same name and signature is created and defined in the sub class. Then the method of the sub class overrides the method of the super class. i.e. when the object of sub class calls this method, the method of the sub class is called instead of the super class. |

|  |
| --- |
| **class A {**  **int i, int j;**  **A(int a, int b) {i=a;j=b;}**  **void show() {System.out.println(i+” ”+j);} }**  **class B extends A { int k;**  **B (int a, int b, int c) { super(a,b); k =c; }**  **void show() {System.out.println(k);}**  **public static void main(String args[]) {**  **B subOb = new B (1,2,3);**  **subOb.show(); // Calls the subOb’s show method which was overridden }}** |

**Run Time Polymorphism and Dynamic Dispatch**

|  |
| --- |
| Call to an overridden method is resolved at run time   * In own words, suppose a class has members and method. The sub class inheriting from the super class has the same method (overridden method). Now we define a super class object and a sub class object. Now when we create a reference variable of super class and then assign the object of the sub class, then at run time the compiler determines at run time that the sub class method needs to be called.   Concept = Reference variable of a super class can be assigned the object of a sub class. This causes run time polymorphism. |

Example

|  |
| --- |
| **class A {**  **void callme() {**  **System.out.println(“Inside A’s callme method”);}}**  **class B extends A {// Call me is overridden herevoid callme() { System.out.println(“Inside B’s callme method”);}}**  **class C extends A {// Callme is overridden here toovoid callme() {System.out.println(“Inside C’s callme method”);**  **public static void main (String args[]) {A r1 = new A(); r1.show();**  **// A’s method is called**  **A r2 = new B();r2.sow(); // B’s method is called // Run time polymorphismA r3 = new C();r3.show(); // C ‘s method is called// Run time polymorphism}}** |

**Abstract Class and Abstract Method**

|  |
| --- |
| * Abstract class cannot be instantiated but can be sub classed. * Many situations where a super class has only the generalized form and the sub classes define the methods. Example Figure class containing parameters as l, b, h and area function and triangle extending Figure class and defining the area. * If we require a method be overridden by the sub classes, then we add abstract key word to it. * If more than one abstract method is present in a class then the class need to have keyword called abstract * An abstract class cannot be instantiated. |

**Example:**

|  |
| --- |
| **abstract class Figure {**  **abstract area(); }**  **public class Triangle extends Figure {**  **int l,b,h;**  **void area(){**  **System.out.println(l\*b\*h); }**  **public static void main(String args[]) {**  **Triangle tri1 = new Triangle();**  **Tri1.area(); // This prints area of triangle overridden from Figure class. }** |

**Interfaces**

|  |
| --- |
| 1) Interface cannot be instantiated.  2) Interface is a list of public method declarations.  3) When the classes implement the interface, it is class’s responsibility to implement all the methods  4) None of the methods in the interface are defined.  5) A sub-class is allowed to implement multiple interfaces.  5.1) Interface has no state. It means that variables are treated as final static. The implementing class need not have the same variables. If required, then switch to abstract class. |

**Example**

|  |
| --- |
| **public interface Figure {**  **public void area();**  **public void perimeter(); }**  **public class Rectangle implements Figure {**  **int l, int b;**  **public void area() { int area = l \*b;}**  **public void perimeter() {int peri = 2\*(l+b); } }**  **public class Square implements Figure {**  **int s; public void area () {int area = s \* s;}**  **public void perimeter() {int perimeter = 4\*s} }** |

**Garbage Collection**

|  |
| --- |
| * The problem is that the programs create objects which use a sizeable memory space and other resources. Now when the objects are no longer in use they fill up the space and create memory and other resource constraints. * Garbage collector works this way. It remembers all the variables belonging to the program and the objects which are pointed by these variables. These are called Reachable objects. Unreachable objects are those that are not pointed by any of the current programs. * **The garbage collector keeps track of the unreachable objects and deletes them and clears the memory space.** * **The GC acts in the background and cannot be controlled.** * **Package and Import** * **Package is like a directory. It contains group of classes and sub directories or sub packages.** * It can be used to group classes with common purpose together and we can impart access modifiers to the package so that the classes and objects outside the package cannot access classes inside the package. * Import = If we need to use a public class present in another package we need to import the contents of the package into the current class and then call the respective class for inheritance or other purposes. * Stack Trace * The JVM stores the functions in LIFO format as stack at run time. When the main method is executed then main becomes bottom of stack and then any object called, the constructor takes the position and then any other method called takes the position above the stack. * Now, when any method shows abnormal behavior, then error report is created known as stack trace. This is useful for debugging of the code. * Exception Handling * Java exception Handling is managed via five key words = try catch finally throw throws. * Exception handling means error handling. Error due to programmer or due to the system happening at the run time. * Exception Process = When the exception happens, an object representing that exception is created and thrown in the method that caused the error. The method may catch it or throw back. At some point exception is caught and then processed. |

**Example**

|  |
| --- |
| class Exc0 {  public static void main(String args[]) {  int d=0; int a = 42/d;  // We are attempting to divide by zero. }}  When the program is run we get the following output:-   * lang.ArithmeticException: Divide by zero   at Exc0.main(Exc0.java:4) |

|  |
| --- |
| * Reading the compiler message, it is a Arithmetic Exception; When this exception is found by the compiler interrupting the normal behavior or flow of process, then object of exception is created and at once handled by the Default Handler provided by Java Run Time system which displays a string displaying the type of exception and the line of the code where the exception had occurred |

.

**Using try and Catch**

|  |
| --- |
| **class Exc0 {**  **public static void main (String args[]) {**  **int d,a;**  **try { // Monitoring the block of code for Arithmetic Exception d =0; a = 42/d;**  **System.out.println(“This line is not printed.”); }**  **catch (ArithmeticException e) {**  **System.out.println(“Division by Zero”); } }}** |

**Multiple Catch Clauses**

|  |
| --- |
| **class MultiCatch {**  **public static void main (String args[]) {**  **try { // Manually monitoring the block for**  **int a = Integer.parseInt(args[0]);**  **int b = 42/a; }**  **catch (ArithmeticException ae ) {**  **System.out.println(“Divide by zero”); }**  **catch(ArrayOutOBoundsException e) {**  **System.out.println(“No Command line argument present.”); }**  **catch (NumberFormatException e) { System.out.println(“Command Line argument is not a number.”); } }}** |

**Finally Keyword:-**

|  |
| --- |
| You can attach a finally-clause to a try-catch block. The code inside the finally clause will always be executed, even if an exception is thrown from within the try or catch block. If your code has a return statement inside the try or catch block, the code inside the finally-block will get executed before returning from the method. Here is how a finally clause looks: |

|  |
| --- |
| **public void openFile(){**  **FileReader reader = null;**  **try {**  **reader = new FileReader("someFile");**  **int i = reader.read(); }**  **} catch (IOException e) {**  **System.out.println(“IOException. Reader not reading.”);**  **} finally {**  **if(reader != null){**  **try {**  **reader.close(); }**  **catch (IOException e) {**  **System.out.println(“IO Exception. File not closing.”); } }**  **System.out.println("--- File End ---"); } }** |

**TABLE OF CONTENTS –JAVA - Course - DAY 2 – Sunil Kumar Gunasekara**n

|  |
| --- |
| 1) Reading from Excel File  2) Date Object to String Conversion  3) String Builders  4) Exception Handling |

1. **Reading from Excel File:-**

|  |
| --- |
| **public class ReadExcelFile {**  **public void readExcel() throws BiffException, IOException {**  **String FilePath = "D:\\sampledoc.xls";**  **FileInputStream fs = new FileInputStream(FilePath);**  **Workbook wb = Workbook.getWorkbook(fs);// TO get the access to the sheetSheet sh = wb.getSheet("Sheet1");// To get the number of rows present in sheet**  **int totalNoOfRows = sh.getRows();// To get the number of columns present in sheet**  **int totalNoOfCols = sh.getColumns();**  **for (int row = 0; row < totalNoOfRows; row++) {**  **for (int col = 0; col < totalNoOfCols; col++) {**  **System.out.print(sh.getCell(col, row).getContents() + "\t"); }**  **System.out.println(); } }**  **public static void main(String args[]) throws BiffException, IOException {**  **ReadExcelFile DT = new ReadExcelFile(); DT.readExcel(); } }**  **OutputUsername password testuser1 testpassword1 testuser2 testpassword2 testuser3 testpassword3 testuser4 testpassword4Output:system oracle** |

**Enum Example**

|  |
| --- |
| **class EnumExample1{**  **public enum Season { WINTER, SPRING, SUMMER, FALL }**  **public static void main(String[] args) {**  **for (Season s : Season.values())**  **System.out.println(s); }}**  **Output:WINTERSPRINGSUMMERFALL** |

**Enum with specific values inside a class:-**

|  |
| --- |
| **class EnumExample4{**  **enum Season{ WINTER(5), SPRING(10), SUMMER(15), FALL(20);**  **private int value;**  **private Season(int value){ this.value=value; } }**  **public static void main(String args[]){**  **for (Season s : Season.values()) System.out.println(s+" "+s.value); }}**  **Output:WINTER 5SPRING 10SUMMER 15FALL 20** |

**Date Object to String conversion**

|  |  |  |
| --- | --- | --- |
| Letter | Description | Examples |
| y | Year | 2013 |
| M | Month in year | July, 07, 7 |
| d | Day in month | 1-31 |
| E | Day name in week | Friday, Sunday |
| a | Am/pm marker | AM, PM |
| H | Hour in day | 0-23 |
| H | Hour in am/pm | 1-12 |
| m | Minute in hour | 0-60 |
| s | Second in minute | 0-60 |

**Note**  
 **For complete date and time patterns, please refer to this** [java.text.SimpleDateFormat JavaDoc](http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html)

1. **Date Example**

If ‘M’ is 3 or more, then the month is interpreted as text, else number

1. **Date = 7-Jun-2013**

|  |
| --- |
| **SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy"); String dateInString = "7-Jun-2013";**  **try { Date date = formatter.parse(dateInString); System.out.println(date);**  **System.out.println(formatter.format(date));**  **} catch (ParseException e) { e.printStackTrace(); }**  **OutputFri Jun 07 00:00:00 MYT 201307-Jun-2013** |

1. **Date = 07/06/2013**

|  |
| --- |
| **SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy"); String dateInString = "07/06/2013"; try { Date date = formatter.parse(dateInString); System.out.println(date); System.out.println(formatter.format(date)); } catch (ParseException e) { e.printStackTrace(); }OutputFri Jun 07 00:00:00 MYT 201307/06/2013** |

1. **Date = Jun 7, 2013**

|  |
| --- |
| **SimpleDateFormat formatter = new SimpleDateFormat("MMM dd, yyyy"); String dateInString = "Jun 7, 2013"; try { Date date = formatter.parse(dateInString); System.out.println(date); System.out.println(formatter.format(date)); } catch (ParseException e) { e.printStackTrace(); }OutputFri Jun 07 00:00:00 MYT 2013Jun 07, 2013** |

1. **Date = Fri, June 7 2013**

|  |
| --- |
| **SimpleDateFormat formatter = new SimpleDateFormat("E, MMM dd yyyy"); String dateInString = "Fri, June 7 2013"; try { Date date = formatter.parse(dateInString); System.out.println(date); System.out.println(formatter.format(date)); } catch (ParseException e) { e.printStackTrace(); }OutputFri Jun 07 00:00:00 MYT 2013Fri, Jun 07 2013** |

1. **Date and Time Example**
2. Date and Time = **Friday, Jun 7, 2013 12:10:56 PM**

|  |
| --- |
| **SimpleDateFormat formatter = new SimpleDateFormat("EEEE, MMM dd, yyyy HH:mm:ss a"); String dateInString = "Friday, Jun 7, 2013 12:10:56 PM"; try { Date date = formatter.parse(dateInString); System.out.println(date); System.out.println(formatter.format(date)); } catch (ParseException e) { e.printStackTrace(); }OutputFri Jun 07 12:10:56 MYT 2013Friday, Jun 07, 2013 12:10:56 PM** |

**String Builder**

|  |
| --- |
| **public class StringBuilderClass {**  **final static String filename = "C:\\Users\\user\\Dropbox\\JOB INTERVIEW FOLDER\\JAVA" + "**[**\\entire-java-code-repo\\FileReaderTextFile.txt**](file:///\\entire-java-code-repo\\FileReaderTextFile.txt)**";**  **Logger logger = Logger.getLogger(StringBuilderClass.class);**  **public static void main(String[] args) {**  **// StringBuilder with 16 empty elements**  **StringBuilder sb = new StringBuilder();**  **sb.append("Hello from JCG");**  **System.out.println("sb appends a string: " + sb);**  **// append a character char c = '!';**  **sb.append(c);**  **System.out.println("sb after appending a char: " + sb);**  **sb.insert(6, "everyone ");**  **System.out.println("sb after insert: " + sb);**  **// StringBulder with a initialized capacity**  **StringBuilder sbnew = new StringBuilder(15);**  **sbnew.append(123456789);**  **System.out.println("sb with length " + sbnew.length() + " and capacity " + sbnew.capacity() + " appends an int: " + sbnew);**  **// delete 234**  **sbnew.delete(1, 4);**  **System.out.println("sb after delete: " + sbnew);**  **// read from a file and append into a StringBuilder every new line try { BufferedReader br = new BufferedReader(new FileReader(filename)); StringBuilder sbFile = new StringBuilder(); String line = br.readLine(); while (line != null) { // append the line of the file sbFile.append(line); // separate the line with a '@' sbFile.append('@'); // read the next line of the file line = br.readLine(); } // this string contains the character sequence String readFile = sbFile.toString(); br.close(); System.out.println("from file: " + readFile); } catch (FileNotFoundException e) { e.printStackTrace(); } catch (IOException e) { e.printStackTrace(); } }}**  **Outputsb appends a string: Hello from JCGsb after appending a char: Hello from JCG!sb after insert: Hello everyone from JCG!sb with length 9 and capacity 15 appends an int: 1234567sb after delete: 1567from file: first line; @second line; @third line; @fourth line; @fifth line;@sixth line; @seventh line; @** |

**Sample Page Object method**

|  |
| --- |
| @Overridepublic void waitToBeDisplayed(long waitTime) throws RequirementsException |

**JAVA – DAY – 3 –COURSE- SUNIL KUMAR GUNASEKARAN**

|  |
| --- |
| * A) Collections – Integer Array, ArrayList, linked list, hash set, hash map * B) Data structures – Linear Search, Binary search * C) Sorts – Bubble and Merge Sort * D) Serialization and de-serialization * E) Object mapper |

**COLLECTIONS**

**Integer Array Declaration and Sizing**

|  |
| --- |
| **int check[] = {1,1,2,3,5,8};**  **n = check.length // checking the size.**  **int fibo[] = new int[10];**  **fibo[0] = 1; // filling values**  **fibo[1] = 1;**  **m = fibo.length; // checking the size.** |

**ArrayList Methods - Ordered List**

|  |
| --- |
| **ArrayList<String> arraylist = new ArrayList<String>();**  **arrayList.add(“JAVA”);**  **arrayList.add(2,”C++”);**  **arrayList.isEmpty() // verify if arrayList is empty**  **arrayList.indexOf(“C++”) // return index of element**  **arrayList.size() // Size of arrayListsArrayList can be read using IteratorIterator<String> arrayIterator = arrayList.iterator();**  **ArrayList<String> cloneArrayList = (ArrayList<String>)**  **arrayList.clone(); // Clone arraylist**  **arryaList1.addAll(arrayList2); // Add elements of array2 into array1**  **subList1 = arrayList1.subList(2,4); // Sub list of array list.**  **//Iterator for loop**  **for (String stringElement: arrayList)System.out.println(stringElement);** |

**Iterator**

|  |
| --- |
| **List<String> myList = new ArrayList<String>();**  **myList.add(“Java”);**  **myList.add(“Unix”);**  **Iterator<String> itr = myList.iterator();**  **while(itr.hasNext()) {**  **System.out.println(itr.next());}**  **Iterator itr = list1.iterator();itr.hasNext() – Navigation**  **itr.next() – return element**  **itr.remove() – remove element** |

**LinkedList - Unordered list- Sunil Kumar Gunasekaran**

|  |
| --- |
| **LinkedList<String> L1 = new LinkedList<String>();**  **L1.add(“Orange”);L1.add(“Apple”);**  **//Methods**  **L1.size();L1.isEmpty();**  **L1.contains(“element”) – Read elements via iterator.**  **linkedList.addAll(arrayList); - Add elements of arraylist to Linked ListlinkedList.clear() – delete all elements in linked listlinkedListIterator.remove() - Individual delete** |

**HashSet (Unordered, Unique list)**

|  |
| --- |
| **HashSet<String> HS1 = new HashSet<String>();**  **HS1.add(“first”);HS1.remove(“first”);Use iterator to navigate through hashSetIterator <String> HSItr1= HS1.iterator();while (HSItr1.hasNext()) {HSItr1.next();} HS1.addAll(HS2); // Add all elements from another hash setHS1.clear() // Clear all elements of hash set** |

**HashMap - (unsorted key value pair)**

|  |
| --- |
| **HashMap<String, String> hashMap = new HashMap<String, String>();**  **hashMap.put(“first”,”FIRST INSERTED”);**  **hashMap.put(“second”, “SECOND INSERTED”);**  **hashMap.put(“third”, “THIRD INSERTED”);**  **hashMap.get(“first”); // get value of keyhashMap.size(); - size of hashMaphashMap.isEmpty(); // boolean of emtpy array** |

* **DATA STRUCTURES**

**Linear Search Return location**

|  |
| --- |
| **O(n)Speed of iteration grows**  **public static int linearSearch(int[] arrayList,int searchValue) {**  **int size = arrayList.length;for (int i=0; i< size; i++) { if(arrayList[i]==searchValue)return i;}** |

**Binary Search Return location for sorted array**

**log(n) Speed of iteration**

|  |
| --- |
| **public int binarySearch(ArrayList<Integer> arrayList, searchValue)**  **{{int start =0;int end = arrayList.size() -1;**  **while (start<=end) {int mid = (start+ end)/2;**  **if (searchValue == arrayList.get(mid))return mid;**  **if (searchValue > arrayList.get(mid))start = mid + 1;**  **if (searchValue < arrayList.get(mid)) end = mid -1; }return -1;}** |

**Bubble Sort n^2 – Speed of computation**

|  |
| --- |
| **public static void bubble\_srt( int a[], int n ){ int i, j,t=0;**  **for (i = 0; i < n; i++) {**  **// since highest value is put at the last in first iteration**  **for (j = 1; j < n-i; j++) {**  **if(a[j-1] > a[j]) {**  **t = a[j-1];**  **a[j-1]=a[j];**  **a[j]=t; } } } }// end of bubble\_srt()** |

**Merge Sort n log n – Fastest computation speed Gravity Sort**

|  |
| --- |
| **TopDownMergeSort(A[], B[], n){**  **CopyArray(A, 0, n, B);**  **TopDownSplitMerge(B, 0, n, A);**  **}**  **TopDownSplitMerge(B[], begin, end, A[]){**  **if(end - begin < 2) return;**  **middle = (end + begin) / 2;**  **TopDownSplitMerge(A, begin, middle, B);**  **TopDownSplitMerge(A, middle, end, B);**  **TopDownMerge(B, begin, middle, end, A);}TopDownMerge(A[], begin, middle, end, B[]) { i = begin, j = middle; for (k = begin; k < end; k++) { if (i < middle && (j >= end || A[i] <= A[j])) { B[k] = A[i]; i = i + 1; } else { B[k] = A[j]; j = j + 1; }}**  **}CopyArray(A[], begin, end, B[]){ for(k = begin; k < end; k++) B[k] = A[k];}** |

**JAVA - SERIALIZATION**

|  |
| --- |
| Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory. |

|  |
| --- |
| **Serializing an Object** The ObjectOutputStream class is used to serialize an Object. T he following SerializeDemo program instantiatesan Employee object and serializes it to a file.When the prog ram is done executing , a file named employee.ser is created. T he prog ram does not g enerateany output, but study the code and try to determine what the prog ram is doing .  Note: When serializing an object to a file, the standard convention in Java is to g ive the file a .ser extension. |

|  |
| --- |
| **import java.io.\*;**  **public class SerializeDemo{**  **public static void main(String [] args){**  **Employee e = new Employee();**  **e.name = "ABD";**  **e.address = "ABD, ABD";**  **e.SSN = 112255;**  **e.number = 101;**  **try{FileOutputStream fileOut =new FileOutputStream("/tmp/employee.ser");**  **ObjectOutputStream out = new ObjectOutputStream(fileOut);**  **out.writeObject(e);**  **out.close();fileOut.close();**  **System.out.printf("Serialized data is saved in /tmp/employee.ser");}catch(IOException i){i.printStackTrace();}}}** |

**Deserializing an Object**

The following DeserializeDemo prog ram deserializes the Employee object created in the SerializeDemoprog ram. Study the prog ram and try to determine its output:

|  |
| --- |
| **import java.io.\*;**  **public class DeserializeDemo{**  **public static void main(String [] args){**  **Employee e = null;**  **try{FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");**  **ObjectInputStream in = new ObjectInputStream(fileIn);**  **e = (Employee) in.readObject();**  **in.close();**  **fileIn.close();}**  **catch(IOException i){**  **i.printStackTrace();**  **return;}**  **catch(ClassNotFoundException c)**  **{System.out.println("Employee class not found");**  **c.printStackTrace();**  **return;}**  **System.out.println("Deserialized Employee...");**  **System.out.println("Name: " + e.name);**  **System.out.println("Address: " + e.address);**  **System.out.println("SSN: " + e.SSN);**  **System.out.println("Number: " + e.number);}}**  **This would produce the following result:**  **Deserialized Employee...Name: ABCDAddress:ABCD, ABCDSSN: 0Number:101Here are following important points to be noted:T he try/catch block tries to catch a ClassNotFoundException, which is declared by the readObject()method. For a JVM to be able to deserialize an object, it must be able to find the** |

**OBJECT MAPPER WITH EXAMPLE** Class Employee with getters and setters.

* JavaObject to JSON Conversion and JSON to JavaObject conversion.

**DAY 4 and DAY 5 – JAVA PROGRAM ASSIGNMENT – Sunil Kumar Gunasekaran**

**JAVA Assignment - Submitted by Sunil Kumar Gunasekaran a) Can you print below diagram?**\*\* \*\* \* \*\* \* \* \*\* \* \*\* \*\*######################################################################

**b) Can you print first 10 prime numbers?**

|  |
| --- |
| public class PrimeNumberCount {  public static void main(String[] args) {  System.out.println("Prime Number List");  System.out.println(" "+ "1");  int flag = 0;  int count = 3;  int primenumbercount =2;  System.out.println("Entering for loop.");    do {  int numberdividebytwo = (count / 2);  for (int i=2; i< count ; i++)  { if (count %i == 0 | (count %2 == 0 && count != 2) | (count %3 == 0 && count !=3) | (count %5 == 0 && count !=5) | (count %7 == 0 && count!=7)) { break; }  flag =1; }    if (flag ==1) {  System.out.println(count);  primenumbercount++;  flag=0;}    if (primenumbercount==10) { break;}    count++;  }while (count<=1000); }    }  **Output:** Prime Number List  1 2  Entering for loop.  3 5 7 11 13 17 19 23  **try {File file = new File("./FirstTenPrimeNumbers.txt");**  **// if file doesnt exists, then create it if (!file.exists()) { file.createNewFile()**;  **c) Write to a file**  **FileWriter fw = new FileWriter(file.getAbsoluteFile());**  **BufferedWriter bw = new BufferedWriter(fw);**  **bw.write("File input line one.");**  **for (int i=0; i< device\_ids.size(); i++)**  **{ bw.write("File input line two."); }**  **bw.close();System.out.println("Done");}**  **catch (IOException e) { e.printStackTrace();**  **}** |

#####################################################################

**c) Three Ways of specifying file directories**

|  |
| --- |
| **public interface Variables {**  **static final String PM\_BUILD\_WORKSPACE = System.getenv("PM\_WORKSPACE");**  **1) String csvFile = TEST\_DATA\_CSV\_FILE; // From system properties**  **2) // String csvFile = "../data/TestData.csv"; // One folder down**  **3) // String csvFile = "./data/TestData.csv"; // Current folder** |

######################################################################

**d) Please implement a function that accepts two parameters say Parameter1 & Parameter2 and gives the output of the number of times the parameter1 occurs in parameter2**

|  |
| --- |
| **// Reading an Integer**  **public static void main (args[]) int a = args[0];**  **// Reading a single string**  **Scanner SC = new Scanner(System.in);**  **System.out.println("Enter parameter1:");**  **String parameter1 = SC.next();// reading a statement**  **BufferedReader reader = new BufferedReader(new InputStreamReader(System.in)); System.out.println("Enter parameter2:");**  **String parameter2= reader.readLine();**  **// Converting parameter2 into scanner object**  **Scanner SCParameter2 = new Scanner(parameter2);**  **// Storing words of parameter2 in a linked list.**  **LinkedList words = new LinkedList();**  **while (SCParameter2.hasNext()) { words.add(SCParameter2.next()); }**  **// Iterating through the linked list and counting number of occurrences of parameter1 in parameter2.**  **Iterator LLIterator = words.iterator();**  **int count =0; while (LLIterator.hasNext()) { if(parameter1.equalsIgnoreCase(LLIterator.next())) { count++; } } System.out.println("The Number of occurances of parameter1 in parameter2 is::" + count);** |

######################################################################**e)**

**Given an array of integers, sort the integer values.**

|  |
| --- |
| **package Assignment;**  **// import java.io.System;**  **public class BubbleSort{public static void main(String args[]) {**  **int i; int array[] = {12,9,4,99,120,1,3,10};// prints the value before sorting array.**  **System.out.println("Values Before bubble sort of Integers:\n");**  **for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" "); System.out.println();**  **// sorting array**  **bubble\_srt(array, array.length);**  **// printing the elements of array after the sort**  **System.out.print("Values after the sort:\n");**  **for(i = 0; i < array.length ; i++)**  **System.out.print(array[i]+" ");**  **System.out.println();}**  **// end of main**  **// static bubble sort method**  **public static void bubble\_srt( int a[], int n )**  **{int i, j,t=0;**  **for (i = 0; i < n; i++) {**  **// since highest value is put at the last in first iteration**  **for (j = 1; j < n-i; j++) {**  **if(a[j-1] > a[j]) { t = a[j-1]; a[j-1]=a[j]; a[j]=t; } } } }**  **// end of bubble\_srt() }// end of class**  **Output Values Before bubble sort of Integers:12 9 4 99 120 1 3 10 Values after the sort: 1 3 4 9 10 12 99 120** |

**####################################################################**

**f) Given an array of integers, print only odd numbers.**

|  |
| --- |
| **package Assignment;**  **public class OddNumbers {**  **public static void main (String args[]) {**  **int i; int array[] = {12,9,4,99,120,1,3,10};**  **// print the elements of array**  **System.out.print("Elements of the array are ::\n");**  **for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" ");**  **System.out.println(); System.out.println();**  **// logic for printing the odd elements of the array System.out.println("Printing the odd numbers of the array::");**  **for (i=0;i < array.length;i++ ) { if (array[i] % 2 != 0 ) { System.out.print(array[i]+" "); } else continue;}**  **Output**  **Elements of the array are 12 9 4 99 120 1 3 10Printing the odd numbers of the array 9 99 1** |

#######################################################################

**3) Given an array of integers move all even numbers to the beginning of the array.**

|  |
| --- |
| **package Assignment;**  **public class MoveEven { public static void main (String args[]) {**  **int i; int array[] = {12,9,4,99,120,1,3,10};// The array elements before moving even elements**  **System.out.println("Values Before moving even integers front of array\n"); for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" ");**  **System.out.println();**  **// Function which moves the even elements to the front of the array. move(array, array.length);**  **// Printing the array elements after the even integers are moved to front. System.out.println("Values After moving even integers front of array\n"); for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" ");**  **System.out.println(); }**  **public static void move (int a[],int n) {**  **int i,j,t; for(i = 0; i < n; i++) {**  **if (a[i]%2 ==0) { for (j = i; j > 0; j--) {**  **t = a[j-1]; a[j-1]=a[j]; a[j]=t; } } } }}**  **Output Values Before moving even integers front of array12 9 4 99 120 1 3 10 Values After moving even integers front of array10 120 4 12 9 99** 1 |

3######################################################################

**4) Print the unique numbers and also print the number of occurrences of duplicate numbers in a two dimensional array.**

|  |
| --- |
| **package Assignment;**  **// This can be accomplished using hash tables. Please look at it.**  **public class Unique {public static void main(String[] args)**  **{int i; int array[] = {12,9,4,99,120,1,3,10, 12, 4,4, 120,3,3,3};**  **// Printing the array elements int limit = array.length; System.out.println("Printing the elements of array\n");**  **for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" ");**  **System.out.println();// initializing a two dimensional array. int record[][]= new int [limit][2];// filling the two dimensional array.**  **for (i=0;i<array.length;i++) {**  **record[i][0] = 0; record[i][1] = 0; }**  **int flag;// For pasting the unique elements into another array.**  **for (i=0;i<array.length;i++) { flag =1; for (int j=0;j<i;j++ )**  **{ if (array[i] == array[j]) { flag ++; } }**  **if (flag == 1) {**  **holder[i][0] = array[i]; } }**  **// For counting the number of occurrences.**  **int j; flag=1; for (i=0;i<holder.length;i++) {**  **flag=0; for (j=i;j<array.length;j++) {**  **if (record[i][0] == array[j]) { flag++; } }**  **// Assigning flag value to the holder.**  **record[i][1]= flag;}**  **// Printing the unique elements and number of their occurrences in 2D array. System.out.println("Printing the unique elements as 2D array");**  **for (i=0;i<record.length;i++) {**  **System.out.println(record[i][0]+" "+record[i][1]);}} }**  **Output Printing the elements of array12 9 4 99 120 1 3 10 12 4 4 120 3 3 3 Printing the unique elements as 2D array 12 2 9 1 4 3 99 1 120 2 1 1 3 4 10** |

1#####################################################################

**5) Given an array of integers check the Fibonacci series.**

|  |
| --- |
| **package Assignment;**  **import java.util.Scanner;**  **public class Fibonnacci {static int n; static int [] fibo;**  **static Scanner console=new Scanner (System.in);**  **public static void main(String[] args) {**  **// Array for testing whether it is fibonacci or not.**  **int check[] = {1,1,2,3,5,8};**  **// Printing the given array elements.**  **System.out.println("Printing the given array::");**  **for (int j=0;j<check.length;j++)**  **{ System.out.print(check[j]+" "); }**  **System.out.println();n = check.length;**  **// Generating new array containing fibonnaci numbers.**  **fibo = new int [n];**  **fill(fibo,n);**  **boolean flag = true;**  **int i; for (i=0;i<n;i++) { if (fibo[i] != check[i]) {**  **flag = false; break; } }**  **if (flag) {**  **System.out.println("The given array elements form fibonnacci series."); } else {**  **System.out.println("The given array elements do not form fibonnacci series."); } }**  **// Logic for generating Fibonnacci numbers.**  **public static void fill(int[]fibo, int n) {**  **int i =0; fibo[0] = 1; fibo[1] = 1;**  **for (i=2;i<n;i++) fibo[i]=fibo[i-1]+fibo[i-2];**  **}// end of fill function**  **}// end of class**  **Output Printing the given array 1 1 2 3 5 8 The given array elements form fibonnacci series.** |

######################################################################

**6) Given an array of integers check the Palindrome of the series**

|  |
| --- |
| **package Assignment;**  **public class Palindrome {**  **public static void main(String[] args) {**  **int i; int array[] = {1,3,4,3,1};**  **// prints the value before sorting array.**  **System.out.println("Elements of the array:");**  **for(i = 0; i < array.length; i++)**  **System.out.print( array[i]+" ");**  **// System.out.println();**  **int flag =1;**  **for (i=0; i< (array.length/2);i++)**  **{ if (array[i] != array[array.length -i-1] )**  **{ flag = 0; } }**  **System.out.println();**  **System.out.println();**  **if (flag == 0) {**  **System.out.println("It is not a palindrome");}**  **else { System.out.println("It is a palindrome."); }}}**  **Output Elements of the array: 1 3 4 3 1Its a palindrome.** |

**7) Given a string print the unique words of the string.**

|  |
| --- |
| **package Assignment;**  **import java.util.HashSet;**  **import java.util.Iterator;**  **import java.util.Scanner;**  **import java.util.Set;**  **// import java.util.TreeSet;**  **// Implementing sets to find the unique words.**  **public class UniqueWord {**  **public static void main(String[] args) {**  **// Hash Set implementing the Set.**  **Set words = new HashSet();**  **// using a sample string to print the unique words**  **String sample ="This is a test is a test a test test";**  **Scanner in = new Scanner(sample);**  **//System.out.println("Please enter the string");**  **while (in.hasNext()) {String word = in.next();**  **// using the add function to add the words into the hash set.**  **words.add(word);}**  **// Used for moving through the set and printing the words.**  **Iterator iter = words.iterator();**  **// Printing the unique words of the string.**  **System.out.println("Printing the unique words of the given string::");**  **for (int i = 1; i <= 20 && iter.hasNext(); i++)**  **// using iterator function to read the elements of hash set. System.out.print(" "+ iter.next()+ " "); } }**  **Output: Printing the unique words of the given string:: is test a This** |

**#######################################################################**

**8) Given a string print the reverse of the string.**

|  |
| --- |
| **package Assignment;**  **// program for printing the reverse of the string.**  **public class StringReverse { public static void main(String[] args) {**  **String str = "molecule"; String reverse ="";**  **int i=0; // printing the original string**  **System.out.println("Original String:: "+ str); // converting string to character array. char rev[] = str.toCharArray();**  **// appending characters to reverse string.**  **for (i=rev.length-1;i>=0;i--) {**  **reverse = reverse + rev[i]; }**  **System.out.println();**  **// Printing the reversed String**  **System.out.println("Reversed String is:: " + reverse); } }**  **Output Original String:: moleculeReversed String is::**  **Elucelom** |

####################################################################

**10) Read a file content and write it to a new file in reverse order.(reverse line 1-10 to line 10-1)**

|  |
| --- |
| **package Assignment;**  **import java.io.\*;**  **import java.util.LinkedList;**  **public class ReverseFile {**  **public static void main(String args[]) {**  **try{// using fileinputstream to read contents inputFile.txt**  **FileInputStream fstream = new FileInputStream("C:\\Users\\Sunil Kumar\\Desktop\\White Box Training\\Java Programs\\Assignment\\inputFile.txt");**  **DataInputStream in = new DataInputStream(fstream);**  **BufferedReader br = new BufferedReader(new InputStreamReader(fstream));**  **String strLine;**  **// using LinkedList to store the lines in the file.**  **LinkedList list = new LinkedList();**  **//Reading input file line by line**  **while ((strLine = br.readLine()) != null) {**  **list.add(strLine); }**  **// Opening the outPut.txt file using FileWriter.**  **FileWriter filestream = new FileWriter("C:\\Users\\Sunil Kumar\\Desktop\\White Box Training\\Java Programs\\Assignment\\outputFile.txt");**  **BufferedWriter out = new BufferedWriter(filestream);**  **// Writing the lines in reverse fashion into outputFile.txt**  **int i; int len = list.size();**  **for (i=len-1;i>=0;i--) {**  **out.write(list.get(i));**  **out.write("\n"); }**  **out.close();**  **in.close(); }**  **catch (Exception e){**  **//Catch exception if any**  **System.err.println("Error: " + e.getMessage()); } }}**  **Input**  **File.txt**  **First line.**  **Second Line.**  **Third Line.**  **Fourth Line.**  **Fifth Line.**  **Sixth Line.**  **Seventh Line.**  **Eighth Line.**  **Ninth Line.**  **Tenth Line.**  **outputFile.txt**  **Tenth Line.**  **Ninth Line.**  **Eighth Line.**  **Seventh Line.**  **Sixth Line.**  **Fifth Line.**  **Fourth Line.**  **Third Line.**  **Second Line.**  **First line.** |

#######################################################################

**1) Write a java program which provides API for database "select" and "Update" package MySQL;**

|  |
| --- |
| **import java.sql.\*;**  **public class DBAccess {**  **public static Statement stmt = null;**  **public static Connection conn = null;**  **public ResultSet execute(String query) {**  **ResultSet rs = null;try {**  **// Execute the query and return the ResultSet**  **stmt.executeQuery(query);**  **rs = stmt.getResultSet(); }**  **catch (Exception e) {**  **System.err.println ("Cannot connect to database server"); System.out.println(e);**  **//e.printStackTrace();**  **}**  **return rs; }**  **public int update(String query) {**  **int count = 0;**  **try {**  **count = s.executeUpdate(query); }**  **catch (Exception e) {**  **System.err.println("Cannot conect to database server."); System.out.println(e); }**  **return count; }**  **public static void main (String[] args) {**  **//Connection conn = null;DBAccess db = new DBAccess();**  **try { String userName = "root";**  **String password = "good";**  **// localhost - Name of the server.**  **String url = "jdbc:mysql://localhost/test";**  **// Create one driver instance and create one or more connection instances.**  **// Standard syntax of creating instance of singleton class.**  **Class.forName ("com.mysql.jdbc.Driver").newInstance ();**  **// Connection instance using the Driver.**  **conn = DriverManager.getConnection (url, userName, password);**  **System.out.println ("Database connection established");**  **stmt = conn.createStatement ();**  **int count;**  **// Two types of methods present in the JDBC code - executeUpdate and // executeQuery**  **// Passing the query and updating the record.**  **String query2 = "Update EMP set email='hare@gmail.com' where id = 2;";**  **count = db.update(query2);**  **System.out.println("Updated record count = " + count);**  **// Passing query and s executing query and returning rs. String query1 = "select \* from EMP";**  **ResultSet rs = db.execute(query1);**  **while (rs.next ()) {**  **int idVal = rs.getInt ("id"); String nameVal = rs.getString ("name");**  **String catVal = rs.getString ("email");**  **System.out.println ( "id = " + idVal + ", name = " + nameVal + ", email = " + catVal); ++count; }**  **rs.close (); s.close ();s.close ();**  **// System.out.println (count + " rows were inserted");}**  **catch (Exception e) {**  **System.err.println ("Cannot connect to database server"); System.out.println(e); e.printStackTrace(); }**  **finally { if (conn != null) { try { conn.close ();**  **System.out.println ("Database connection terminated"); }**  **catch (Exception e) { /\* ignore close errors \*/ } } } } }Output Database connection established Updated record count = 1 id = 1, name = Abcd, email =** [**abcded@gmail.com**](mailto:abcded@gmail.com) **id = 2, name = Manish, email =** [**hare@gmail.com**](mailto:hare@gmail.com) **id = 1, name = Balaji, email =** [**abcded@gmail.com**](mailto:abcded@gmail.com) **Database connection terminated** |



