**Onpassive Assessment**

**==============================================================================**

1. **How to print date in specific format?**

**package** ramesh.pasula.onpassive;

**import** java.text.SimpleDateFormat;

**import** java.time.LocalDateTime;

**import** java.time.format.DateTimeFormatter;

**import** java.util.Date;

**public** **class** DateFormatEx {

**public** **static** **void** main(String[] args) {

Date date = **new** Date();

SimpleDateFormat formatter = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

System.***out***.println(formatter.format(date));

// Java 8 Feature its replacement of SimpleDateFormat

LocalDateTime now = LocalDateTime.*now*();

DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");

System.***out***.println(dtf.format(now));

}

}

1. **How to remove Whitespaces from String**

**package** ramesh.pasula.onpassive;

**public** **class** RemoveWhiteSpaceEx {

**public** **static** **void** main(String[] args) {

String str = "Ramesh Pasula";

// 1. One-Way using predefined

String after\_remove\_white\_space = str.replaceAll("\\s", "");

System.***out***.println(after\_remove\_white\_space);

// 2.Second-Way Custom

**char**[] strArray = str.toCharArray();

StringBuffer stringBuffer = **new** StringBuffer();

**for** (**int** i = 0; i < strArray.length; i++) {

**if** ((strArray[i] != ' ') && (strArray[i] != '\t')) {

stringBuffer.append(strArray[i]);

}

}

String after\_remove\_white\_space1 = stringBuffer.toString();

System.***out***.println(after\_remove\_white\_space1);

}

}

1. **What is difference between Hibernate Session get() and load() method**

Both of these methods are present as part Session and takes 2 parameters. 1: entityClassType and 2: Object (primarykey value).

get() method within the session class will immediately goes to the database and executes the query and fetches the data from database into an object and returns to us. This indicates get supports eager initialization.

Ex:

Branch branch = session.load(Branch.class, 1):

load() method within the session class will not goes to the database, rather it creates an Proxy object ontop of the EntityClass we specified at Runtime and populates the PrimaryKey into the Proxy object and returns to us.

when we call session.load(Branch.class, 1)-> internally load method is creating an proxy class which exactly looks like the original class, by adding database logic in fetching the record of data from database, something similar to below.

class Branch$Proxy extends Branch {

Branch branch; // into this branch object it will store primarykey value we passed to load method

String branchName = branch.getBranchName();

When the user tries to access the data (by calling accessor methods) by using Proxy, then the Proxy object goes to the database and fetches the record of data from the database and returns the value to us. This indicates load() method supports lazy initialization.

Proxy is always an class which acts as a substitue of the original class, we can access the same functionality of the original through proxy as well. When we talk to the proxy, internally proxy goes to the original in serving the original functionality to the end user.