**Assisted Practice: 1.1 Type Casting**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to perform implicit and explicit type casting

This lab has three subsections, namely:

* + 1. Writing a program in Java to implement implicit and explicit type casting
    2. Executing the program and verifying how the conversion of data types happen
    3. Pushing the code to your GitHub repositories

**Step 1.1.1:** Writing a program in Java to implement implicit and explicit type casting

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the below code resolving the warning and errors due compatibility-related issues

**public** **class** typeCasting {

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

//implicit conversion

System.***out***.println("Implicit Type Casting");

**char** a='A';

System.***out***.println("Value of a: "+a);

**int** b=a;

System.***out***.println("Value of b: "+b);

**float** c=a;

System.***out***.println("Value of c: "+c);

**long** d=a;

System.***out***.println("Value of d: "+d);

**double** e=a;

System.***out***.println("Value of e: "+e);

System.***out***.println("\n");

System.***out***.println("Explicit Type Casting");

//explicit conversion

**double** x=45.5;

**int** y=(**int**)x;

System.***out***.println("Value of x: "+x);

System.***out***.println("Value of y: "+y);

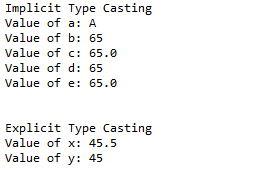
}

}

**Step 1.1.2:** Executing the program and verifying how the conversion of data types happen

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 1.1.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 1.2 Access Modifiers**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to verify the working of access modifiers

This lab has three subsections, namely:

* + 1. Writing a program in Java to implement access modifiers
    2. Executing the program and verifying how the access modifiers work
    3. Pushing the code to your GitHub repositories

**Step 1.2.1:** Writing a program in Java to implement access modifiers

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

//1. Class is having Default access modifier

**class** defAccessSpecifier

{

**void** display()

{

System.***out***.println("You are using defalut access specifier");

}

}

**public** **class** accessSpecifiers1 {

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

//default

System.***out***.println("Dafault Access Specifier");

defAccessSpecifier obj = **new** defAccessSpecifier();

obj.display();

}

}

//2. using private access specifiers

**class** priaccessspecifier

{

**private** **void** display()

{

System.***out***.println("You are using private access specifier");

}

}

**public** **class** accessSpecifiers2 {

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

//private

System.***out***.println("Private Access Specifier");

priaccessspecifier obj = **new** priaccessspecifier();

//trying to access private method of another class

//obj.display();

}

}

//3. using protected access specifiers

**package** pack1;

**public** **class** proaccessspecifiers {

**protected** **void** display()

{

System.***out***.println("This is protected access specifier");

}

}

//create another package

**package** pack2;

**import** pack1.\*;

**public** **class** accessSpecifiers3 **extends** proaccessspecifiers {

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

accessSpecifiers3 obj = **new** accessSpecifiers3 ();

obj.display();

}

}

//4. using public access specifiers

**package** pack1;

**public** **class** pubaccessspecifiers {

**public** **void** display()

{

System.***out***.println("This is Public Access Specifiers");

}

}

//create another package

**package** pack2;

**import** pack1.\*;

**public** **class** accessSpecifiers4 {

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

pubaccessspecifiers obj = **new** pubaccessspecifiers();

obj.display();

}

}

**Step 1.2.2:** Executing the program and verifying how the access modifiers work

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*







**Step 1.2.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.1 Methods**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to create methods

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify implementations of methods and ways of calling a method
    2. Executing the program and verifying working of methods
    3. Pushing the code to your GitHub repositories

**Step 2.1.1:** Writing a program in Java to verify implementations of methods and ways of calling a method

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due to compatibility-related issues

//method demo

**public** **class** methodExecution {

**public** **int** multipynumbers(**int** a,**int** b) {

**int** z=a\*b;

**return** z;

}

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

methodExecution b=**new** methodExecution();

**int** ans= b.multipynumbers(10,3);

System.***out***.println("Multipilcation is :"+ans);

}

//call by value

**public** **class** callMethod {

**int** val=150;

**int** operation(**int** val) {

val =val\*10/100;

**return**(val);

}

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

callMethod d = **new** callMethod();

System.***out***.println("Before operation value of data is "+d.val);

d.operation(100);

System.***out***.println("After operation value of data is "+d.val);

}

}

//method overloading

**public** **class** overloadMethod {

**public** **void** area(**int** b,**int** h)

{

System.***out***.println("Area of Triangle : "+(0.5\*b\*h));

}

**public** **void** area(**int** r)

{

System.***out***.println("Area of Circle : "+(3.14\*r\*r));

}

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

{

overloadMethod ob=**new** overloadMethod();

ob.area(10,12);

ob.area(5);

}

}

**Step 2.1.2:** Executing the program and verify working of methods

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*







**Step 2.1.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.2 Constructors**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to create constructors

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify the implementations of constructor types
    2. Executing the program and verifying constructors
    3. Pushing the code to your GitHub repositories

**Step 2.2.1:** Write a program in Java to verify the implementations of constructor types

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

//default constructor

**class** EmpInfo{

**int** id;

String name;

**void** display() {

System.***out***.println(id+" "+name);

}

}

**public** **class** constructorDemo {

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

EmpInfo emp1=**new** EmpInfo();

EmpInfo emp2=**new** EmpInfo();

emp1.display();

emp2.display();

}

}

//parameterized constructor

**class** Std{

**int** id;

String name;

Std(**int** i,String n)

{

id=i;

name=n;

}

**void** display() {

System.***out***.println(id+" "+name);

}

}

**public** **class** paramConstrDemo {

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

Std std1=**new** Std(2,"Alex");

Std std2=**new** Std(10,"Annie");

std1.display();

std2.display();

}

}

**Step 2.2.2:** Executing the program and verifying the constructors

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*





**Step 2.2.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.3 Collections**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to create collections

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify implementations of collections
    2. Executing the program and verifying it is working
    3. Pushing the code to your GitHub repositories

**Step 2.3.1:** Writing a program in Java to verify implementations of collections

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

**import** java.util.\*;

**public** **class** collectionAssisted {

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

//creating arraylist

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

ArrayList<String> city=**new** ArrayList<String>();

city.add("Bangalore");//

city.add("Delhi");

System.***out***.println(city);

//creating vector

System.***out***.println("\n");

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

Vector<Integer> vec = **new** Vector();

vec.addElement(15);

vec.addElement(30);

System.***out***.println(vec);

//creating linkedlist

System.***out***.println("\n");

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

LinkedList<String> names=**new** LinkedList<String>();

names.add("Alex");

names.add("John");

Iterator<String> itr=names.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

//creating hashset

System.***out***.println("\n");

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

HashSet<Integer> set=**new** HashSet<Integer>();

set.add(101);

set.add(103);

set.add(102);

set.add(104);

System.***out***.println(set);

//creating linkedhashset

System.***out***.println("\n");

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

LinkedHashSet<Integer> set2=**new** LinkedHashSet<Integer>();

set2.add(11);

set2.add(13);

set2.add(12);

set2.add(14);

System.***out***.println(set2);

}

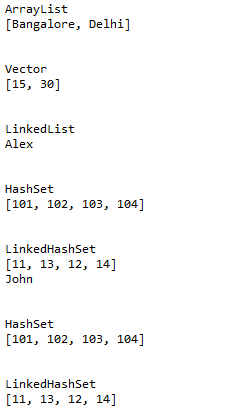
}

}

**Step 2.3.2:** Executing the program and verify it is working

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 2.3.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.4 Map**

This section will guide you to:

* Create a Java project in your IDE
* Write a program in Java to create maps

This lab has three subsections, namely:

2.4.1 Writing a program in Java to verify implementations of maps

2.4.2 Executing the program and verifying it is working

2.4.3 Pushing the code to your GitHub repositories

**Step 2.4.1:** Writing a program in Java to verify implementations of maps

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

**import** java.util.\*;

**public** **class** mapDemo {

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

// map

//Hashmap

HashMap<Integer,String> hm=**new** HashMap<Integer,String>();

hm.put(1,"Tim");

hm.put(2,"Mary");

hm.put(3,"Catie");

System.***out***.println("\nThe elements of Hashmap are ");

**for**(Map.Entry m:hm.entrySet()){

System.***out***.println(m.getKey()+" "+m.getValue());

}

//HashTable

Hashtable<Integer,String> ht=**new** Hashtable<Integer,String>();

ht.put(4,"Ales");

ht.put(5,"Rosy");

ht.put(6,"Jack");

ht.put(7,"John");

System.***out***.println("\nThe elements of HashTable are ");

**for**(Map.Entry n:ht.entrySet()){

System.***out***.println(n.getKey()+" "+n.getValue());

}

//TreeMap

TreeMap<Integer,String> map=**new** TreeMap<Integer,String>();

map.put(8,"Annie");

map.put(9,"Carlotte");

map.put(10,"Catie");

System.***out***.println("\nThe elements of TreeMap are ");

**for**(Map.Entry l:map.entrySet()){

System.***out***.println(l.getKey()+" "+l.getValue());

}

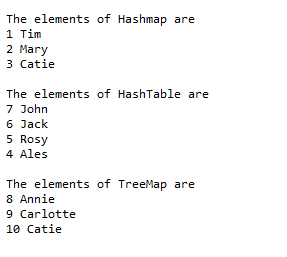
}

}

**Step 2.4.2:** Executing the program and verifying it is working

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 2.4.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.5 Inner Classes**

This section will guide you to:

* Create a Java project in your IDE
* Write a Java program to create inner classes

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify the implementation of inner classes
    2. Executing the program and verifying working of inner classes
    3. Pushing the code to your GitHub repositories

**Step 2.5.1:** Writing a program in Java to verify the implementation of inner classes

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

**public** **class** innerClassAssisted1 {

**private** String msg="Welcome to Java";

**class** Inner{

**void** hello(){System.***out***.println(msg+", Let us start learning Inner Classes");}

}

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

innerClassAssisted1 obj=**new** innerClassAssisted1();

innerClassAssisted1.Inner in=obj.**new** Inner();

in.hello();

}

}

**public** **class** innerClassAssisted2 {

**private** String msg="Inner Classes";

**void** display(){

**class** Inner{

**void** msg(){

System.***out***.println(msg);

}

}

Inner l=**new** Inner();

l.msg();

}

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

innerClassAssisted2 ob=**new** innerClassAssisted2 ();

ob.display();

}

}

//anonymous inner class

**abstract** **class** AnonymousInnerClass {

**public** **abstract** **void** display();

}

**public** **class** innerClassAssisted3 {

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

AnonymousInnerClass i = **new** AnonymousInnerClass() {

**public** **void** display() {

System.***out***.println("Anonymous Inner Class");

}

};

i.display();

}

}

**Step 2.5.2:** Executing the program and verifying the working of inner classes

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*







**Step 2.5.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.6 Strings**

This section will guide you to:

* Create a Java project in your IDE
* Write a Java program to create strings and convert them into StringBuffer and StringBuilder

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify implementations of strings, StringBuffer, and StringBuilder
    2. Executing the program and verifying working of strings
    3. Pushing the code to your GitHub repositories

**Step 2.6.1:** Writing a program in Java to verify implementations strings, StringBuffer, and StringBuilder

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due to compatibility-related issues

**public** **class** stringDemo {

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

//methods of strings

System.***out***.println("Methods of Strings");

String sl=**new** String("Hello World");

System.***out***.println(sl.length());

//substring

String sub=**new** String("Welcome");

System.***out***.println(sub.substring(2));

//String Comparison

String s1="Hello";

String s2="Heldo";

System.***out***.println(s1.compareTo(s2));

//IsEmpty

String s4="";

System.***out***.println(s4.isEmpty());

//toLowerCase

String s5="Hello";

System.***out***.println(s1.toLowerCase());

//replace

String s6="Heldo";

String replace=s2.replace('d', 'l');

System.***out***.println(replace);

//equals

String x="Welcome to Java";

String y="WeLcOmE tO JaVa";

System.***out***.println(x.equals(y));

System.***out***.println("\n");

System.***out***.println("Creating StringBuffer");

//Creating StringBuffer and append method

StringBuffer s=**new** StringBuffer("Welcome to Java!");

s.append("Enjoy your learning");

System.***out***.println(s);

//insert method

s.insert(0, 'w');

System.***out***.println(s);

//replace method

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(0, 2, "hEl");

System.***out***.println(sb);

//delete method

sb.delete(0, 1);

System.***out***.println(sb);

//StringBuilder

System.***out***.println("\n");

System.***out***.println("Creating StringBuilder");

StringBuilder sb1=**new** StringBuilder("Happy");

sb1.append("Learning");

System.***out***.println(sb1);

System.***out***.println(sb1.delete(0, 1));

System.***out***.println(sb1.insert(1, "Welcome"));

System.***out***.println(sb1.reverse());

//conversion

System.***out***.println("\n");

System.***out***.println("Conversion of Strings to StringBuffer and StringBuilder");

String str = "Hello";

// conversion from String object to StringBuffer

StringBuffer sbr = **new** StringBuffer(str);

sbr.reverse();

System.***out***.println("String to StringBuffer");

System.***out***.println(sbr);

// conversion from String object to StringBuilder

StringBuilder sbl = **new** StringBuilder(str);

sbl.append("world");

System.***out***.println("String to StringBuilder");

System.***out***.println(sbl);

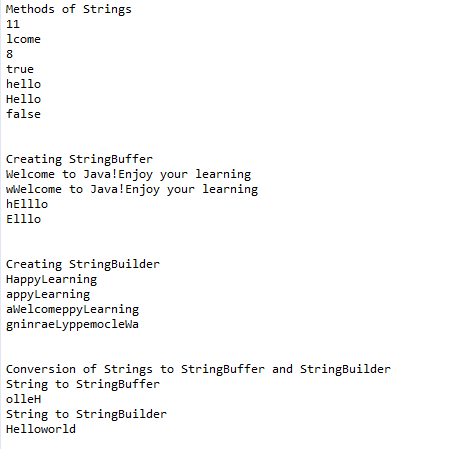
}

}

**Step 2.6.2:** Executing the program and verifying the working of strings

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 2.6.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.7 Arrays**

This section will guide you to:

* Create a Java project in your IDE
* Write a Java program to create arrays

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify implementation of arrays
    2. Executing the program and verifying working of arrays
    3. Pushing the code to your GitHub repositories

**Step 2.7.1:** Writing a program in Java to verify implementation of arrays

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

**public** **class** arrayAssisted {

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

//single-dimensional array

**int** a[]= {10,20,30,40,50};

**for**(**int** i=0;i<5;i++) {

System.***out***.println("Elements of array a: "+a[i]);

}

//multidimensional array

**int**[][] b = {

{2, 4, 6, 8},

{3, 6, 9} };

System.***out***.println("\nLength of row 1: " + b[0].length);

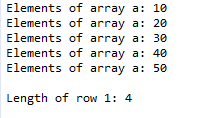
}

}

**Step 2.7.2:** Executing the program and verifying the working of arrays

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 2.7.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**

**Assisted Practice: 2.8 Regular Expressions**

This section will guide you to:

* Create a Java project in your IDE
* Write a Java program to search a specific string from a given set of strings using regular expressions

This lab has three subsections, namely:

* + 1. Writing a program in Java to verify implementations of regular expressions
    2. Executing the program and verifying working of regular expressions
    3. Pushing the code to your GitHub repositories

**Step 2.8.1:** Writing a program in Java to verify implementations of regular expressions

There are two ways you can perform this step; you can create a new Java project, or you can create a new Java class in the existing project. It is preferable to create a new Java class in the existing project but feel free to explore the first option. The steps mentioned below will work once you create a project in Java.

* Open Eclipse
* *[Right click]* on the **src** folder of the project
* Select *New* -> *Java Class* -> Enter the filename (follow camelCasing)
* Execute the code below resolving the warning and errors due compatibility-related issues

**import** java.util.regex.\*;

**public** **class** regularExpnAssisted {

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

String pattern = "[a-z]+";

String check = "Regular Expressions";

Pattern p = Pattern.*compile*(pattern);

Matcher c = p.matcher(check);

**while** (c.find())

System.***out***.println( check.substring( c.start(), c.end() ) );

}

}

**Step 2.8.2:** Executing the program and verifying the working of regular expressions

Before you execute the program, check for syntactical corrections. If no errors are found, follow the steps mentioned below:

* ***[Right click]*** in the program space
* Select *Run As Java Application*



**Step 2.8.3:** Pushing the code to your GitHub repositories

* Open your command prompt and navigate to the folder where you have created your files.

**cd <folder path>**

* Initialize your repository using the following command:

**git init**

* Add all the files to your git repository using the following command:

**git add .**

* Commit the changes using the following command:

**git commit . -m “Changes have been committed.”**

* Push the files to the folder you initially created using the following command:

**git push -u origin master**