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 2.1.1:** 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 code given below resolving the warning and errors due to 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);

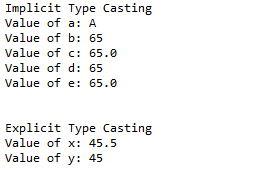
}

}

**Step 2.1.2:** 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 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

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 2.2.1:** 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 2.2.2:** 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 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

3 While Loop

This section will guide you to:

* Understand how to use While Loop

**Development Environment:**

* Java 1.8
* Eclipse

This guide has three subsections, namely:

2.3.1 Creating a java class in eclipse

2.3.2 Executing the program

2.3.3 Pushing the code to your GitHub repositories

**Step 2.3.1:** Creating a java class in eclipse

The while loop in Java is used to iterate a part of the program several times. If the number of iterations is not fixed, it is recommended to use a While Loop.

* Open Eclipse
* Click on File---> Click on New--->Project
* Select the Java project and click on Next
* Enter the project name and click on Finish
* Right click on Project---> New---> Package
* Enter the package name ---> Finish
* Right click on Package---> New---> Class
* Enter the class name---> click on Finish

**Step 2.3.2:** Executing the program

* Write the program for a while loop and click on Save.

**package** AssistedPractice;

**public** **class** WhileLoop1

{

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

{

int i=1;

**while**(i<=10){

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

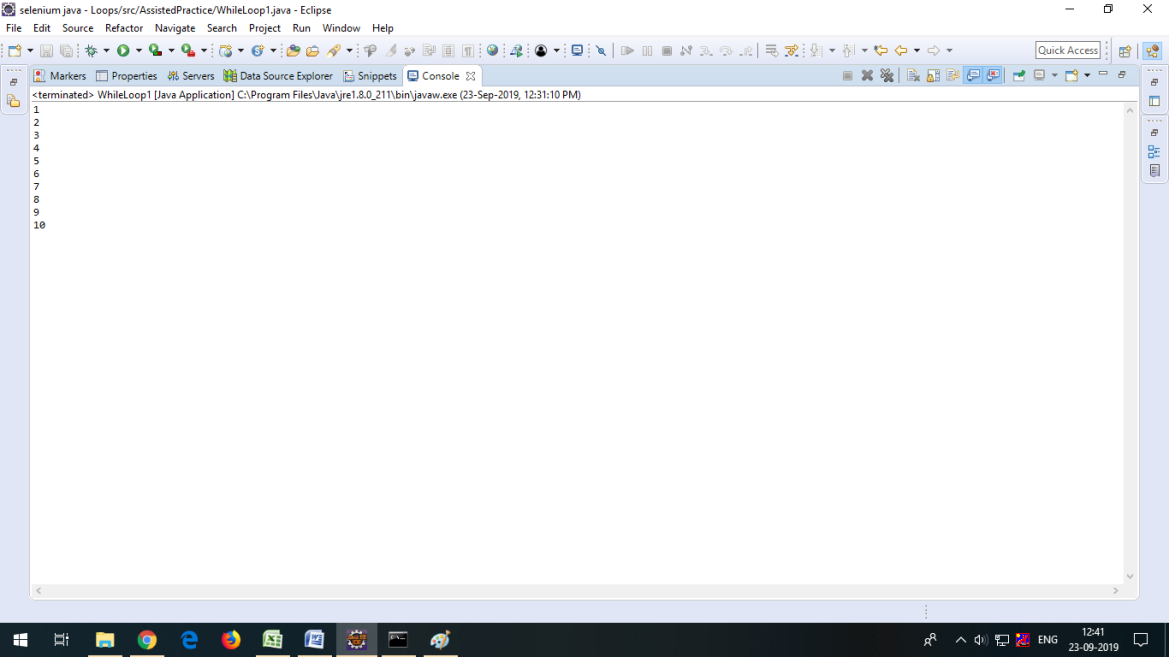
i++;

}

}

}

* Click on Run and check the output in the console.

****

**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

4 Do-While Loop

This section will guide you to:

* Understand how to use Do-While Loop

**Development Environment:**

* Java 1.8
* Eclipse

This guide has three subsections, namely:

2.4.1 Creating a java class in eclipse

2.4.2 Executing the program

2.4.3 Pushing the code to your GitHub repositories

**Step 2.4.1:** Creating a java class in eclipse

The Do-While Loop in Java is used to iterate a part of the program several times. If the number of iterations is not fixed and you must have to execute the loop at least once, it is recommended to use Do-While Loop.

The Do-While loop in Java is executed at least once because the specified condition is checked after the loop body.

* Open Eclipse
* Click on File---> Click on New--->Project
* Select the Java project and click on Next
* Enter the project name and click on Finish
* Right click on Project---> New---> Package
* Enter the package name ---> Finish
* Right click on Package---> New---> Class
* Enter the class name---> click on Finish

**Step 2.4.2:** Executing the program

* Write the program for Do-While Loop and click on Save.

**public** **class** DoWhileExample

{

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

{

int i=1;

**do**

{

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

i++;

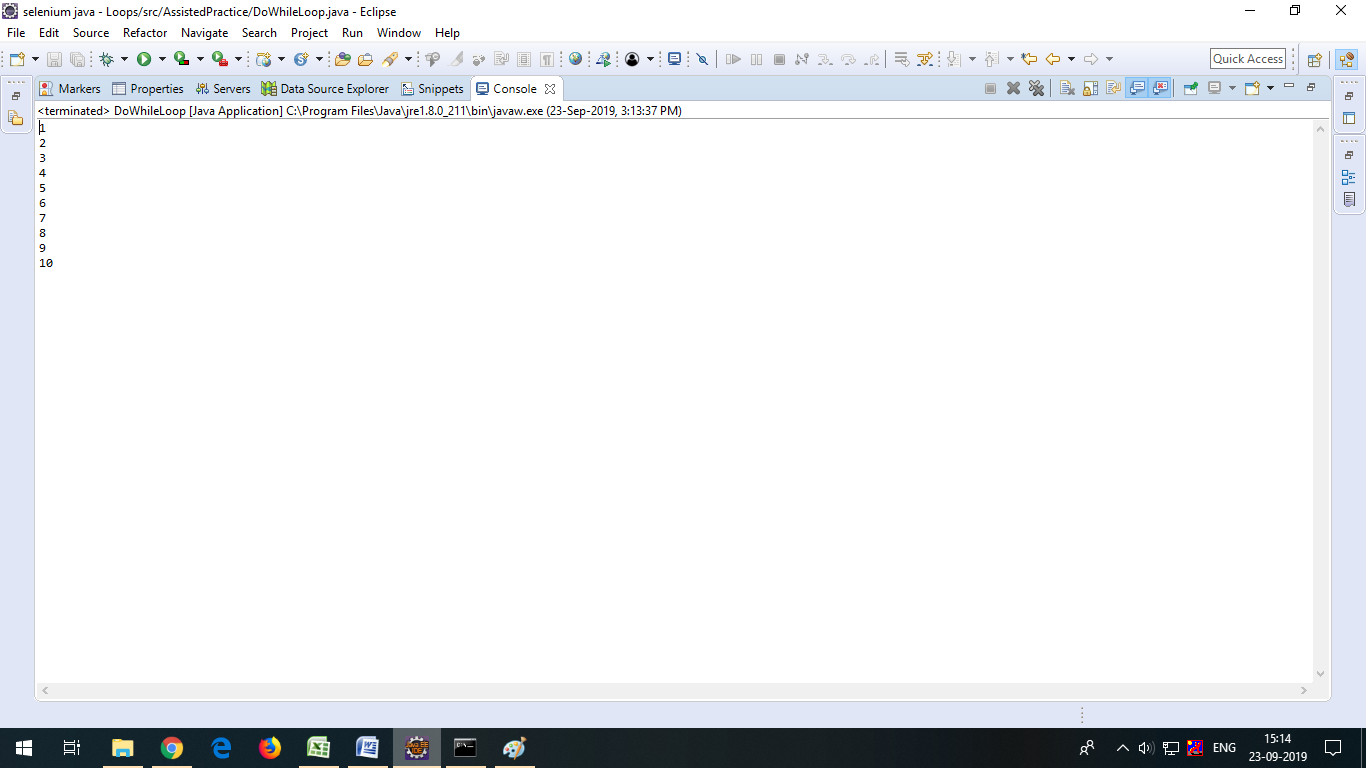
}

**while**(i<=10);

}

}

* Click on Run and check the output in console.

****

**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

5 For Loop

This section will guide you to:

* Understand how to use For loop

**Development Environment:**

* Java 1.8
* Eclipse

This guide has three subsections, namely:

2.5.1 Creating a java class in eclipse

2.5.2 Executing the program

2.5.3 Pushing the code to your GitHub repositories

**Step 2.5.1:** Creating a java class in eclipse

1. For Loop: For Loop is used when the number of iterations is fixed and the user knows exactly how many times the block of code has to be executed.
2. Enhanced For Loop: The Enhanced For Loop is used to traverse array or collection in java. It is easier to use than a simple For Loop because we don't need to increment value and use subscript notation.

It works on elements based on the index. It returns elements one by one in the defined variable.

* Open Eclipse
* Click on File---> Click on New--->Project
* Select the Java project and click on Next
* Enter the project name and click on Finish
* Right click on Project---> New---> Package
* Enter the package name ---> Finish
* Right click on Package---> New---> Class
* Enter the class name---> click on Finish

**Step 2.5.2:** Executing the program

* Program for For Loop

**package** AssistedPractice;

**public** **class** ForLoop

{

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

{

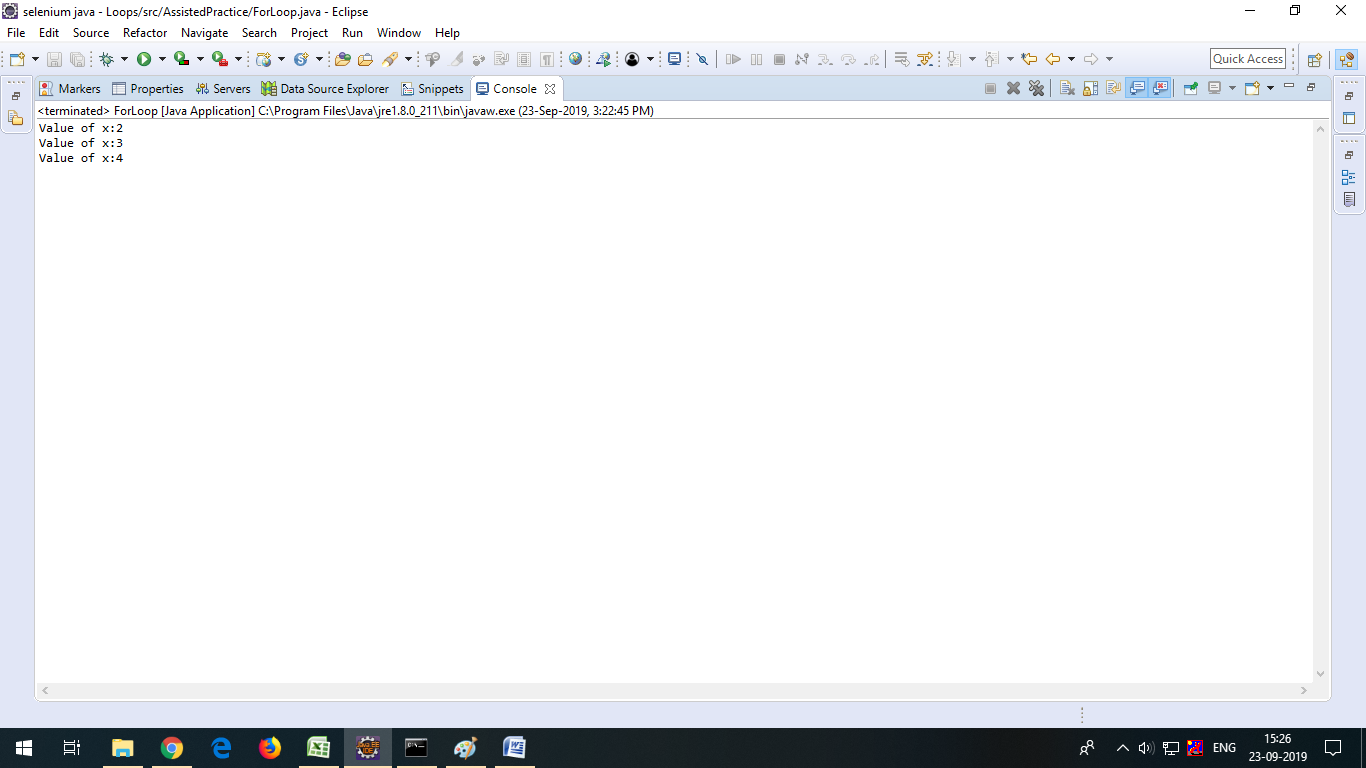
**for** (int x = 2; x <= 4; x++)

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

}

}

* Click on Save--->Click on Run and check the output in console.

****

* Program for Enhanced For Loop.

**package** AssistedPractice;

**public** **class** EnhancedForLoop

{

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

{

int arr[]={12,23,44,56,78};

**for**(int i:arr)

{

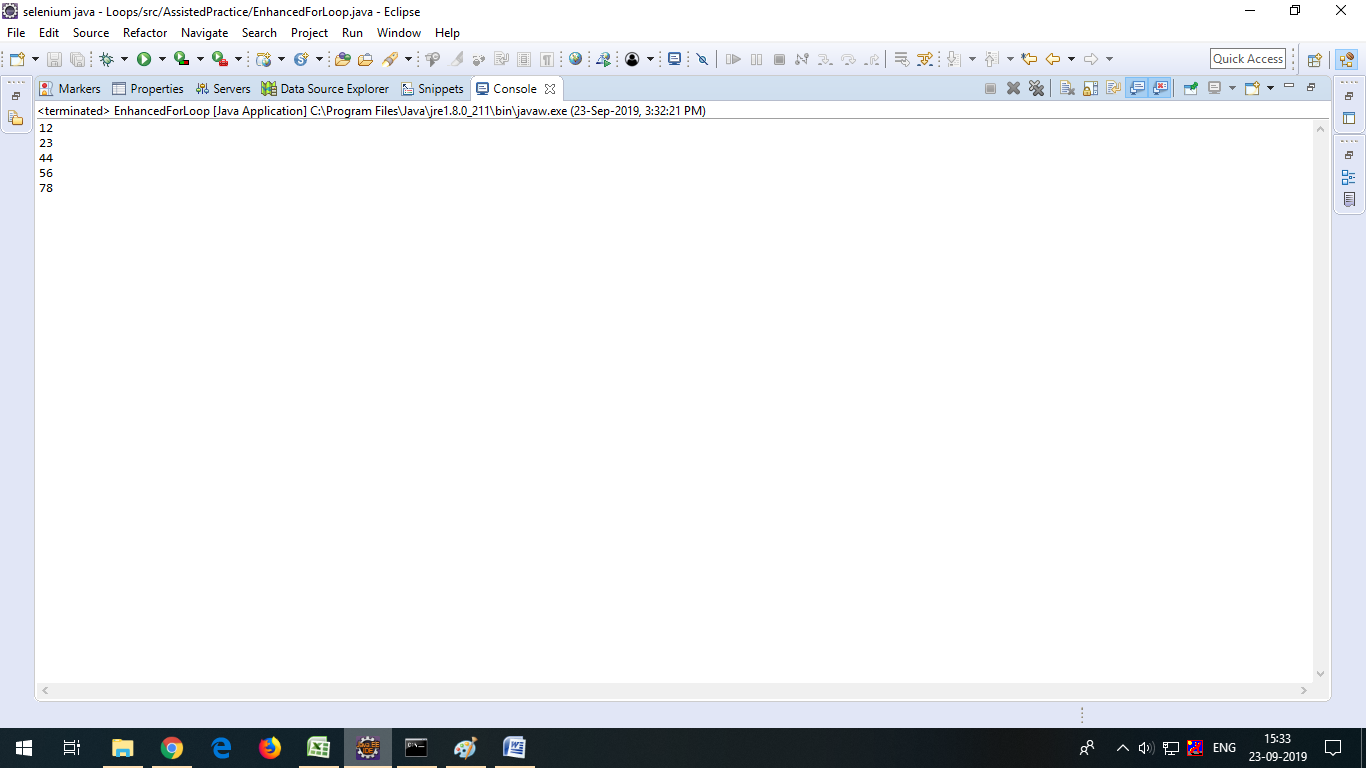
**System**.out.println(i);

}

}

}

* Click on Save--->Click on Run and check the output in console.

****

**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

6 Classes, Objects, and Constructors

This section will guide you to understand:

* What is a Class?
* What is an Object?
* What is a Constructor?
* How to use Class, Object, and Constructor?

**Development Environment:**

* Eclipse
* Java 1.8

This guide has four subsections, namely:

2.6.1 Creating a class

2.6.2 Creating an object

2.6.3 Creating a constructor

2.6.4 Pushing the code to your GitHub repositories

**Step 2.6.1:** Creating a class

* A class is a blueprint from which individual objects are created.

**public** **class** Dog {

**String** breed;

int age;

**String** color;

void barking() {

}

void hungry() {

}

void sleeping() {

}}

**Step 2.6.2:** Creating an object

* If we consider the real world, we can find many objects around us, such as cars, dogs, humans, and so on. All these objects have a state and behavior.
* If we consider a dog, then its state is: name, breed, color, and the behavior is: barking, wagging the tail, and running.
* If you compare the software object with a real-world object, they have very similar characteristics.
* Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods.
* So, in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

**public** **class** Puppy {

**public** Puppy(**String** name) {

// This constructor has one parameter, name.

**System**.out.println("Passed Name is :" + name );

}

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

// Following statement would create an object myPuppy

Puppy myPuppy = **new** Puppy( "tommy" );

}}

**Step 2.6.3:** Creating a constructor

* When discussing about classes, one of the most important subtopics would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.
* Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

**public** **class** Puppy {

**public** Puppy() {

}

**public** Puppy(**String** name) {

// This constructor has one parameter, name.

}}

**Step 2.6.4:** 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

7 Inheritance

This section will guide you to understand:

* What is Inheritance?
* Types of Inheritance in Java
* Why use Inheritance?
* Why is Multiple Inheritance not supported?

**Development Environment:**

* Eclipse
* Java 1.8

This guide has two subsections, namely:

2.7.1 Demonstrate types of inheritance

2.7.2 Push the code to your GitHub repositories

**Step 2.7.1:** Demonstrate types of inheritance

* There are various types of inheritance in Java:

1. Single Inheritance:

In Single Inheritance, one class extends another class (one class only).

Class A{

**public** void methodA()

{

**System**.out.println("Base class method");

}}

Class B **extends** A{

**public** void methodB()

{

**System**.out.println("Child class method");

}

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

{

B obj = **new** B();

obj.methodA(); //calling super class method

obj.methodB(); //calling local method

}

}

1. Multiple Inheritance:

In Multiple Inheritance, one class extends more than one class. Java does not support multiple inheritance.

1. Multi-level Inheritance:

In Multi-level Inheritance, one class can inherit properties from a derived class. Hence, the derived class becomes the base class for the new class.

Class X{

**public** void methodX()

{

**System**.out.println("Class X method");

}

}

Class Y **extends** X{

**public** void methodY(){

**System**.out.println("class Y method");

}

}

Class Z **extends** Y{

**public** void methodZ()

{

**System**.out.println("class Z method");

}

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

{

Z obj = **new** Z();

obj.methodX(); //calling grand parent class method

obj.methodY(); //calling parent class method

obj.methodZ(); //calling local method

}

}

1. Hierarchical Inheritance:

In Hierarchical Inheritance, one class is inherited by many sub classes.

1. Hybrid Inheritance:

Hybrid inheritance is a combination of single and multiple inheritance.

**Note:**Java doesn't support hybrid/multiple inheritance.

**Step 2.13.5:** 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

8 Collections and Generics

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 2.8.1:** 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);

}

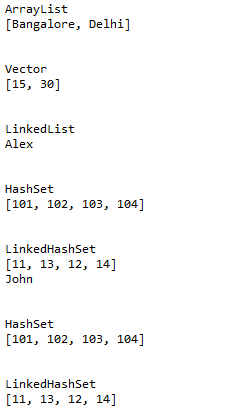
}

}

**Step 2.8.2:** Executing the program and verifying whether 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 2.8.3:** Pushing the codes 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

9 Basic Try-Catch Block

This section will guide you to:

* Implement a try-catch block

This guide has two subsections, namely:

## 2.9.1Writing a try-catch block

2.9.2 Pushing the code to your GitHub repository

## **Step 2.9.1:** Writing a try-catch block

* Write the code given below in the main Java class:

class Example1 {

public static void main(String args[]) {

int num1, num2;

try {

number1 = 0;

System.out.println("Sending the Exception");

number2 = 62 / num1;

System.out.println(num2);

}

  catch (ArithmeticException e) {

/\* This block will only execute if any Arithmetic exception

\* occurs in try block

\*/

System.out.println("We can’t divide any number by zero");

}

catch (Exception e) {

/\* This is a generic Exception handler which means it can handle

/\* all the exceptions. This will execute if the exception is not

\* handled by previous catch blocks.

\*/

System.out.println("Exception occurred");

}

System.out.println("Try-Catch ended.");

}

}

Output:

Sending the Exception

We can’t divide any number by zero

Try-Catch ended.

**Step 2.9.2:** 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

10 Throw and Throws

This section will guide you to:

* Implement **throw** and **throws** keywords

This guide has three subsections, namely:

2.10.1 Writing code for **throw** keyword

2.10.2 Writing code for **throws** keyword

2.10.3 Pushing the code to your GitHub repository

**Step 2.10.1:** Writing code for **throw** keyword

* Write the code given below in a Java file and run it as a Java application:

**public class** TestThrow1 {

**static void** validate(**int** age){

**if**(age<18)

**throw new** ArithmeticException("not valid");

**else**

System.***out***.println("welcome to vote");

}

**public staticvoid** main(String args[]){

*validate*(13);

System.***out***.println("rest of the code...");

}

}

Output:

Exception in thread "main" java.lang.ArithmeticException: not valid

at java1.TestThrow1.validate(TestThrow1.java:7)

at java1.TestThrow1.main(TestThrow1.java:12)

**Step 2.10.2:** Writing code for **throws** keyword

* Write the code given below in a Java file and run it as a Java application:

**import**java.io.IOException;

**class** Testthrows2{

**void** m()**throws**IOException{

**throw new** IOException("device error");//checked exception

}

**void** n() **throws** IOException{

m();

}

**void** p(){

**try**{

n();

}

**catch**(Exception e)

{

System.***out***.println("exception handled");

}

}

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

Testthrows2 obj=**new** Testthrows2();

obj.p();

System.***out***.println("normal flow...");

}

}

Output:

exception handled

normal flow...

**Step 2.10.3:** 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

11 Implementing Try-With Parameter in Java

This section will guide you to:

* Implement a try-with parameter

This guide has two subsections, namely:

2.11.1 Writing code for a try block with parameters

2.11.2 Pushing the code to your GitHub repository

**Step 2.11.1:** Writing code for a try block with parameters

* + Write the code below in a Java file and run it as a Java application:

**import** java.io.FileOutputStream;

**public class** TrywithResouces {

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

**try**(FileOutputStream fileOutputStream =**new** FileOutputStream("abc.txt")){

String msg = "Welcome to java!";

**byte** byteArray[] = msg.getBytes(); //converting string into byte array

fileOutputStream.write(byteArray);

System.***out***.println("Message written to file successfuly!");

}**catch**(Exception exception){

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

}

}

}

Output:

Message written to file successfuly!

declared resources.

**Step 2.11.2:** 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

12 Multiple Catch-With Exception Class Hierarchy

This section will guide you to:

* Implement multiple catch-with exception class hierarchy

This guide has three subsections, namely:

2.12.1 Creating a new Java project

2.12.2 Writing code for multiple catch blocks

2.12.3 Pushing the code to your GitHub repository

**Step 2.12.1:** Creating a new Java project

* Open Eclipse
* Click on File->New->Java Project from the menu bar
* Give the project name as **MultipleCatch** and click **OK**
* Right-click on your project->click on New->click on Class and provide a class name as **multipleCatch** and click **OK**

# Step 2.12.2: Writing code for multiple catch blocks

* Write the code given below in a Java file and run it as a Java application:

package **io.com**;

**public class** MultipleCatchBlock {

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

        try{

**int** a[]=new **int**[5];

            System.out.println(a[10]);

        }

        catch(**ArithmeticException** e){

            System.out.println("Arithmetic exception");

        }

        catch(**ArrayIndexOutOfBoundsException** e){

           System.out.println("ArrayIndexOutOfBounds exception");

        }

        catch(**Exception** e){

            System.out.println("Parent exception");

        }

        System.out.println("rest of the code");

    }

}

**Output:**

Arithmetic Exception

Rest of the code

* Write the code given below in a Java file for a nested try-catch block:

**class** Excep1{

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

**try**{

**try**{

System.***out***.println("going to divide");

**int**b =39/0;

}**catch**(ArithmeticException e){System.***out***.println(e);}

**try**{

**int** a[]=**newint**[5];

a[5]=4;

}

**catch**(ArrayIndexOutOfBoundsException e)

{

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

}

System.***out***.println("another statement");

}

**catch**(Exception e)

{

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

}

System.***out***.println("normal flow..");

}

}

**Output:**

going to divide

java.lang.ArithmeticException: / by zero

java.lang.ArrayIndexOutOfBoundsException: 5

another statement

normal flow..

**Step 2.12.3:** 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

12 Finally Block

This section will guide you to:

* Implement **finally{}** block

This guide has two subsections, namely:

2.13.1 Writing code for **finally{}** block

2.13.2 Pushing code to GitHub repository

**Step 2.13.1:** Writing code for **finally{}** block

### Case 1: Without Exception

* Write the code below in a Java file and run it as a Java application:

**class**TestFinallyBlock{

**publicstaticvoid** main(String args[]){

**try**{

**int** number=25/5;

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

}

**catch**(NullPointerException e)

{

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

}

**finally**

{

System.***out***.println("The Execution of final block always happen ");

}

System.***out***.println("after final the rest of the code....");

}

}

* Output:

5

The Execution of final block always happen

after final the rest of the code....the code...

### Case 2: With Exception

* Write the code below in a Java file and run it as a Java application**:**

**publicclass** TestFinallyBlock1 {

**publicstaticvoid** main(String args[]){

**try**{

**int** number=5/0;

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

}

**catch**(NullPointerException e)

{

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

}

**finally**

{

System.***out***.println("finally block is always executed");

}

System.***out***.println("then rest of the code...");

}

}

Output:

finally block is always executed

Exception in thread "main" java.lang.ArithmeticException: / by zeroat java1.TestFinallyBlock1.main(TestFinallyBlock1.java:7)

### Case 3: Exception handled

* Write the code below in a Java file and run it as a Java application:

**publicclass** TestFinallyBlock2{

**publicstaticvoid** main(String args[]){

**try**{

**int** number=25/0;

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

}

**catch**(ArithmeticException e)

{

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

}

**finally**

{

System.***out***.println("finally block is always executed");

}

System.***out***.println("rest of the code...");

}

}

* Output:

java.lang.ArithmeticException: / by zero

finally block is always executed

rest of the code...

**Step 2.13.2:** 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