**JAVA 2 ASSIGNMENT**

**4099 - RAGHAV GUPTA**

**-------------------------------------------------------------**

**[all the class file are also there in the repository]**

**1. Create Java classes having suitable attributes for Library management system. Use OOPs concepts in your design. Also try to use interfaces and abstract classes.**

**package** exercise2;

**class** Book{

**static int** *count*;

**static**{

*count*=0;

}

**int id**;

String **name**;

String **author**;

**int shelf**;

**boolean issued**;

Book(){

}

Book(String name, String author, **int** shelf){

**this**.**id** = ++*count*;

**this**.**name** = name;

**this**.**author** = author;

**this**.**shelf** = shelf;

**issued** = **false**;

}

**void** showDetails(){

System.***out***.println(**"----- BOOK -----"**);

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

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

System.***out***.println(**"author - "** + **author**);

System.***out***.println(**"shelf - "** + **shelf**);

System.***out***.println(**"issued - "** + **issued**);

}

}

**class** Member{

**static int** *count*;

**int id**;

String **name**;

**static** {

*count*=0;

}

*// we can add here that what all books are currently issued by this member.*

*// Book []issuedBooks = new Book[10];*

Member(String name){

**id** = ++*count*;

**this**.**name** = name;

}

}

**class** Library{

**int numberOfBooks**;

String **librarian**;

**int numberOfMembers**;

Book []**bank**;

Library(){

**bank** = **new** Book[1000];

}

**private void** setLibrarian(String name)

{

**this**.**librarian** = name;

}

**public void** setDetails(**int** books, String name, **int** members){

**numberOfBooks** = books;

**numberOfMembers** = members;

setLibrarian(name);

}

**public void** showDetails(){

System.***out***.println(**"this library is run by - "** + **librarian**);

System.***out***.println(**"its has "** + **numberOfBooks** + **" books."**);

System.***out***.println(**"its is used by "** + **numberOfMembers** + **" members."**);

}

**public void** addBook(Book newBook){

*// to add a new book to the library*

}

**public void** issueBook(Member m){

*// to issue a book by a person*

}

**public void** returnBook(Member m, Book b){

*// to return a book*

}

**public** Book getBookById(**int** id){

*// to search a book by its id.*

}

}

**public class** LibraryDemo {

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

}

}

----- on github, u can see the classes by these names .

**2. WAP to sorting string without using string Methods?.**

**package** exercise2;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public class** SortString {

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

String s = **"raghav"**;

*// convert string to char array*

**char** arr[] = s.toCharArray();

*// sort the char array*

**for**(**int** i=0; i<arr.**length**-1; i++){

**char** min = arr[i];

**int** minIndex = i;

**for**(**int** j=i+1; j<arr.**length**; j++){

**if**(arr[j]<arr[i]){

minIndex = j;

}

}

**char** temp;

temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

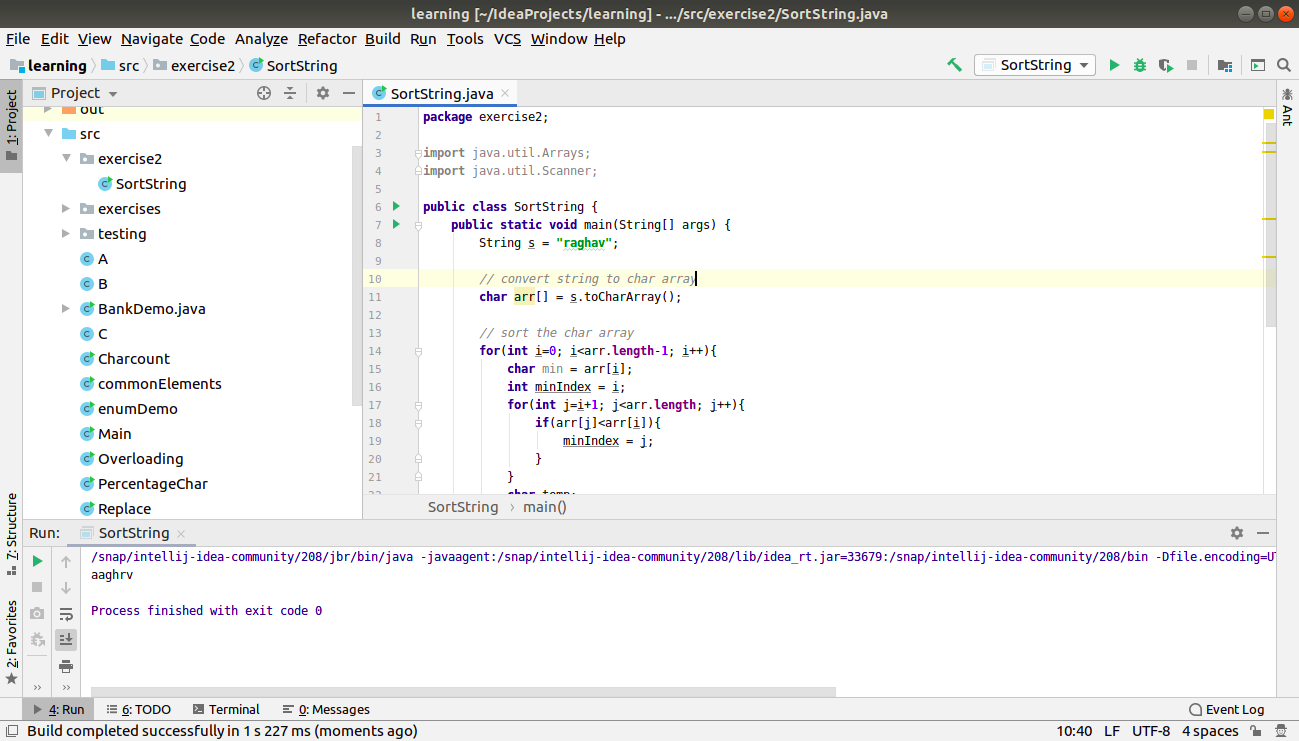
*// convert array back to string*

s = String.*valueOf*(arr);

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

}

}



**3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.**

**ClassNotFoundException**

**ClassNotFoundException** occurs when you try to load a class at runtime using **Class.forName()** or **loadClass()** methods and requested classes are not found in classpath. Most of the time this exception will occur when you try to run application without updating classpath with JAR files.

**package** exercise2;

**public class** classExceptions {

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

**try**{

Class.*forName*(**"Tothenew"**);

}

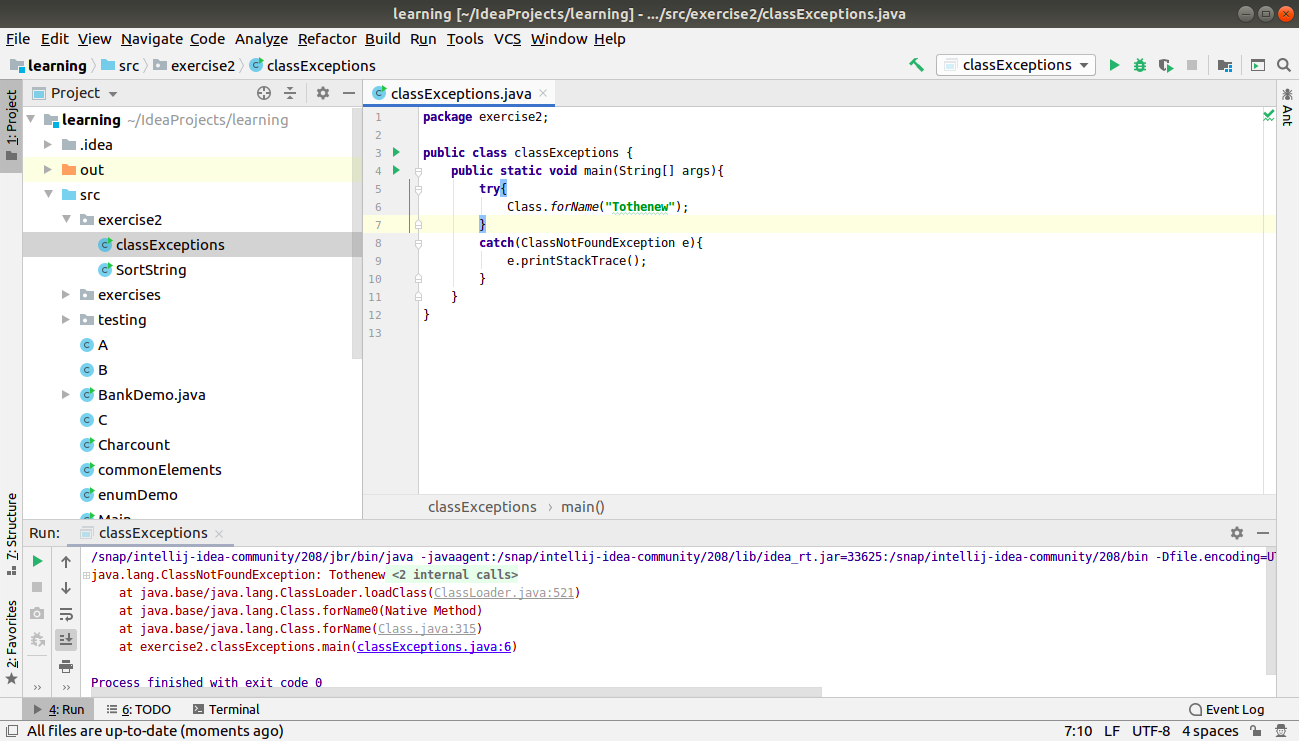
**catch**(ClassNotFoundException e){

e.printStackTrace();

}

}

}



**NoClassDefFoundError**

NoClassDefFoundError occurs when class was present during compile time and program was compiled and linked successfully but class was **not** present during runtime. It is error which is derived from **LinkageError**. Linkage error occurs when a class has some dependencies on another class and latter class changes after compilation of former class.

class A

{

void sayHello()

{

System.out.println("hello!");

}

}

class Ttn {

public static void main(String args[])

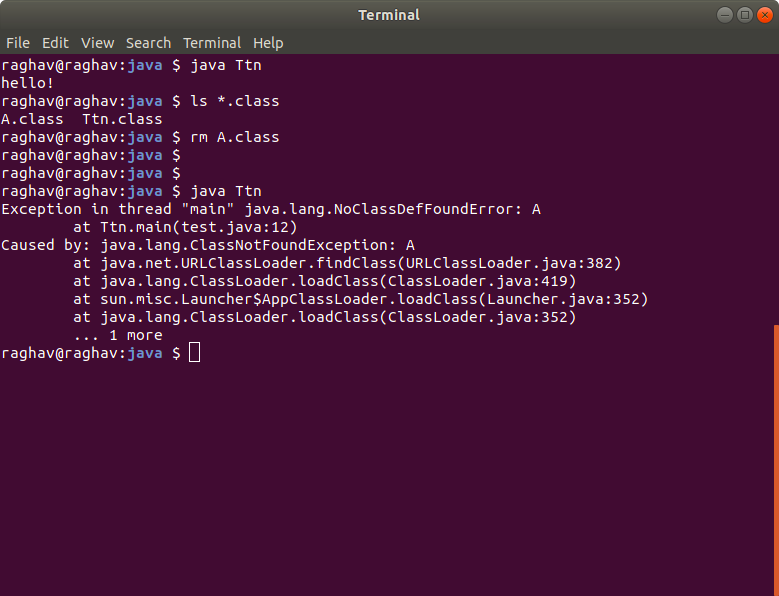
{

A raghav = new A();

raghav.sayHello();

}

}



**4. WAP to create singleton class.**

**package** exercise2;

**public class** Singleton {

**public static** Singleton *single\_instance* = **null**;

**public** String **s**;

**private** Singleton(){

**s** = **"raghav is an awwesome guy"**;

}

**public static** Singleton getInstance(){

**if**(*single\_instance* == **null**)

*single\_instance* = **new** Singleton();

**return** *single\_instance*;

}

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

Singleton a = Singleton.*getInstance*();

Singleton b = Singleton.*getInstance*();

Singleton c = Singleton.*getInstance*();

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

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

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

a.**s** = a.**s**.toUpperCase();

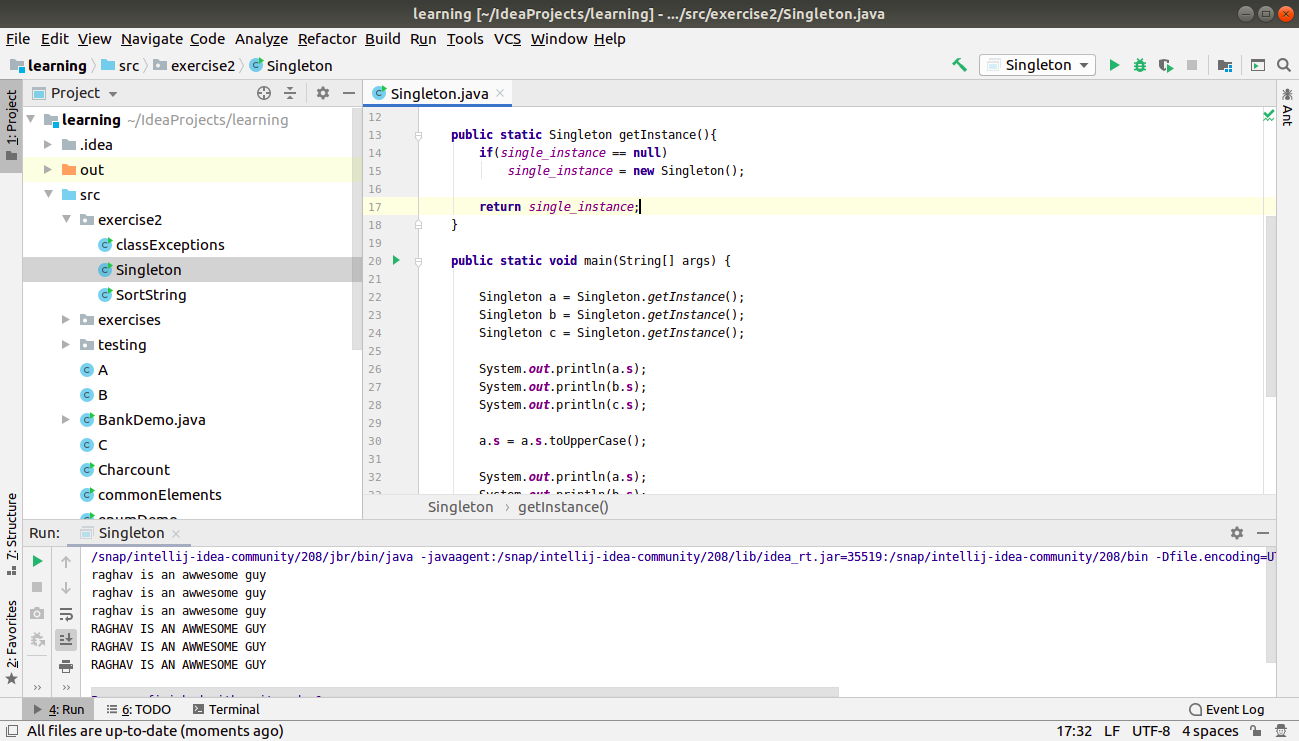
System.***out***.println(a.**s**);

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

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

}

}



We can also do this by making a method with same name as the clas name.

**5. WAP to show object cloning in java using cloneable and copy constructor both.**

**package** exercise2;

**class** Test{

**public int x**,**y**;

}

**class** Test2 **implements** Cloneable{

**int a**, **b**;

Test **t** = **new** Test();

**public** Object clone() **throws** CloneNotSupportedException{

Test2 c = (Test2)**super**.clone();

c.**t** = **new** Test();

**return** c;

}

}

**public class** Cloning {

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

Test2 t1 = **new** Test2();

t1.**a** = 10;

t1.**b** = 20;

t1.**t**.**x** = 100;

t1.**t**.**y** = 200;

Test2 t2 = (Test2)t1.clone();

t2.**a** = 1000;

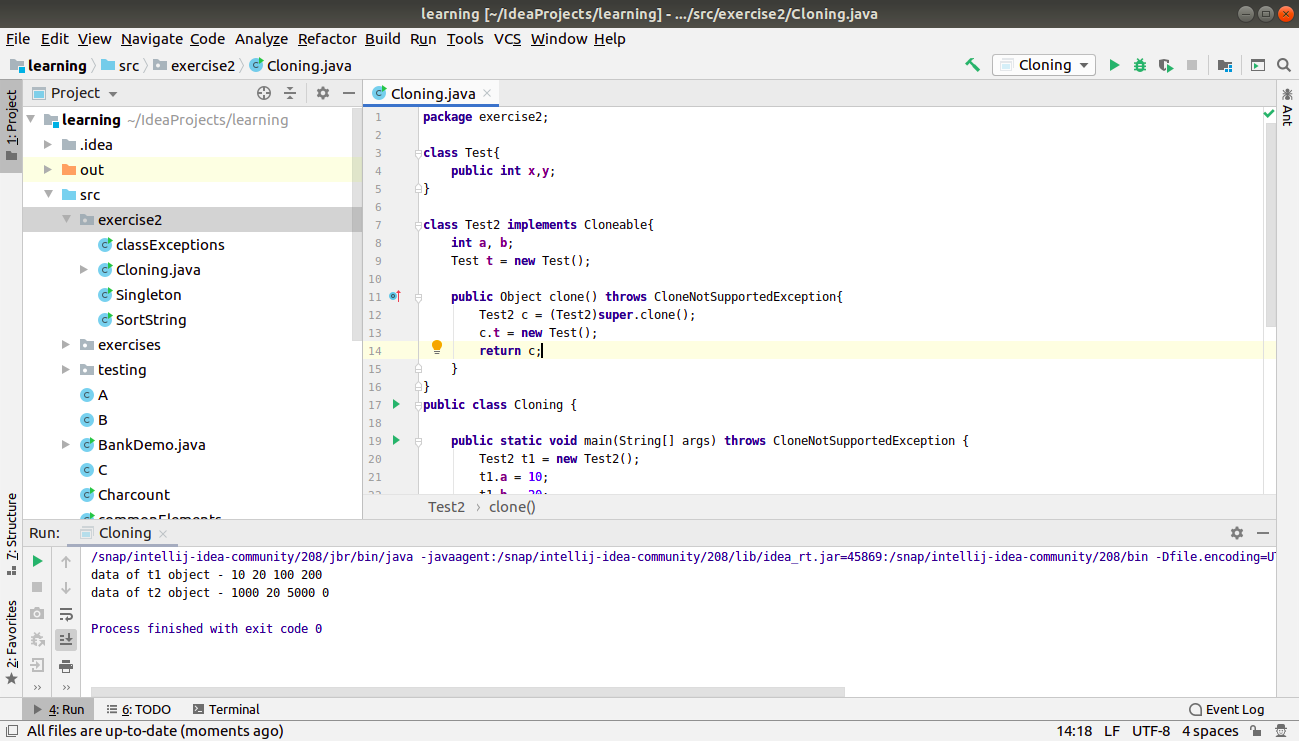
t2.**t**.**x** = 5000;

System.***out***.println(**"data of t1 object - "** + t1.**a** + **" "** + t1.**b** + **" "** + t1.**t**.**x** + **" "** + t1.**t**.**y**);

System.***out***.println(**"data of t2 object - "** + t2.**a** + **" "** + t2.**b** + **" "** + t2.**t**.**x** + **" "** + t2.**t**.**y**);

}

}



**By using copy constructor**

**package** exercise2;

**class** Test3{

**public int a**, **b**;

**public** Test **t**;

Test3(){

**t** = **new** Test();

}

Test3(Test3 temp){

**this**.**a** = temp.**a**;

**this**.**b** = temp.**b**;

**this**.**t** = **new** Test();

**t**.**x** = temp.**t**.**x**;

**t**.**y** = temp.**t**.**y**;

}

}

**public class** CopyConstructor {

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

*// set the values of t1 object*

Test3 t1 = **new** Test3();

t1.**a** = 5;

t1.**b** = 10;

t1.**t**.**x** = 35;

t1.**t**.**y** = 25;

*// copy this object into another object*

Test3 t2 = **new** Test3(t1);

*// modify a property of t2*

t2.**t**.**x** = 5000;

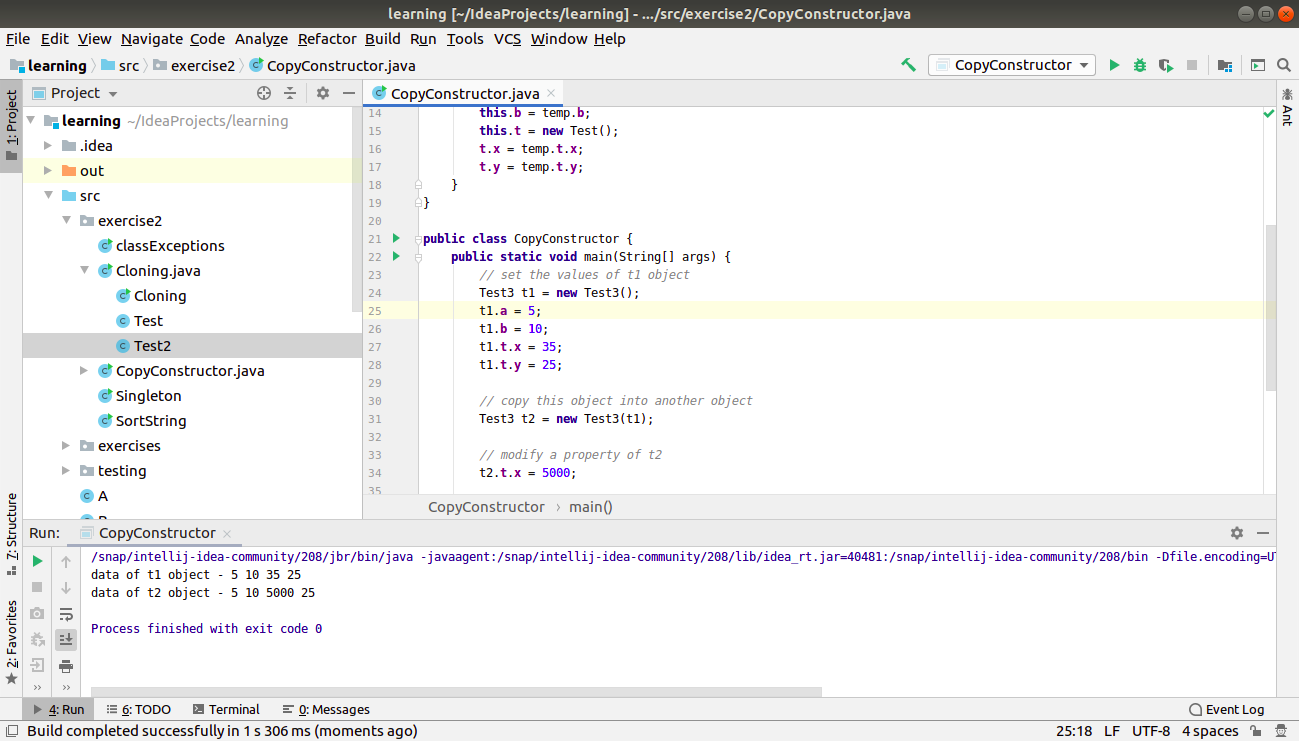
*// view both the objects to very deep copy*

System.***out***.println(**"data of t1 object - "** + t1.**a** + **" "** + t1.**b** + **" "** + t1.**t**.**x** + **" "** + t1.**t**.**y**);

System.***out***.println(**"data of t2 object - "** + t2.**a** + **" "** + t2.**b** + **" "** + t2.**t**.**x** + **" "** + t2.**t**.**y**);

}

}



**6. WAP showing try, multi-catch and finally blocks.**

**package** exercise2;

**public class** TryCatchDemo {

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

**int** a=5;

**int** arr[] = {1,2,3,4,5};

**try**{

*// int c = a/0;*

System.***out***.println(arr[7]); *// to understand better*

}

**catch**(ArithmeticException ae){

System.***out***.println(**"caught inside arithmetic exception specific catch"**);

ae.printStackTrace();

}

**catch**(Exception e){

System.***out***.println(**"caught inside generic catchh"**);

e.printStackTrace();

}

**finally**{

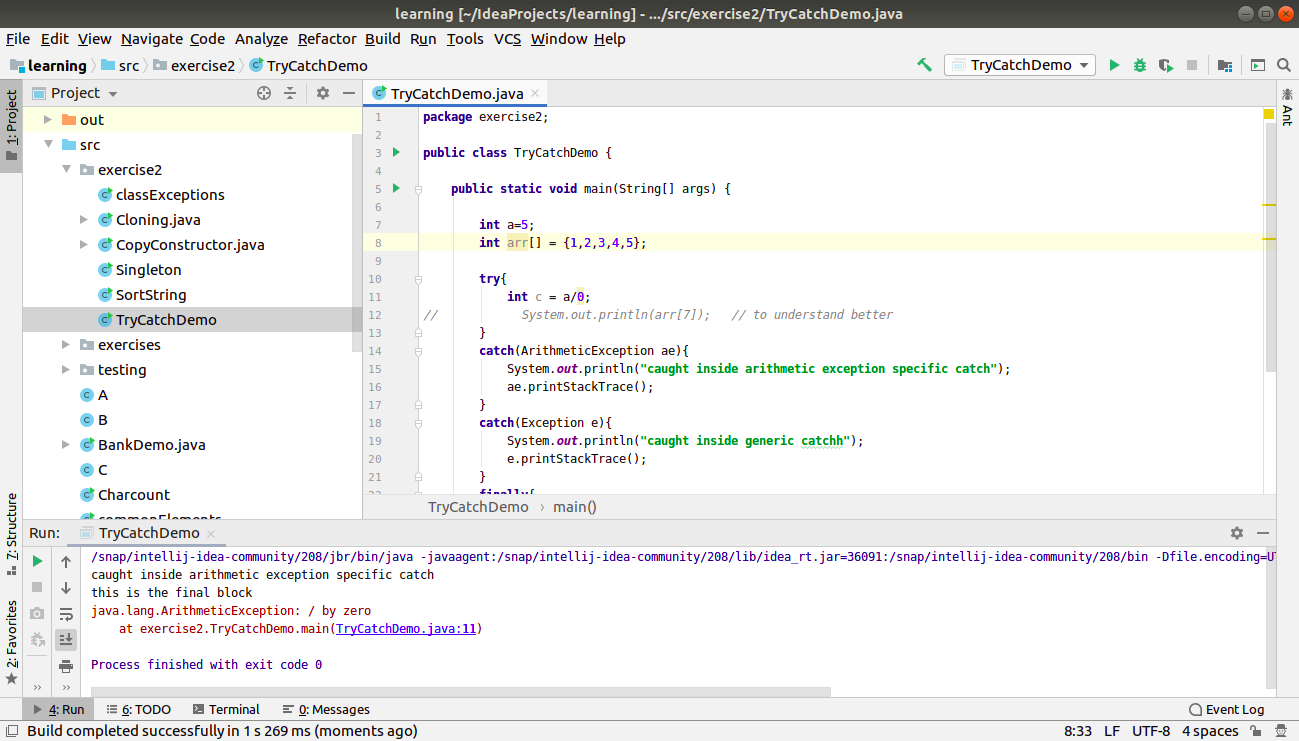
System.***out***.println(**"this is the final block"**);

}

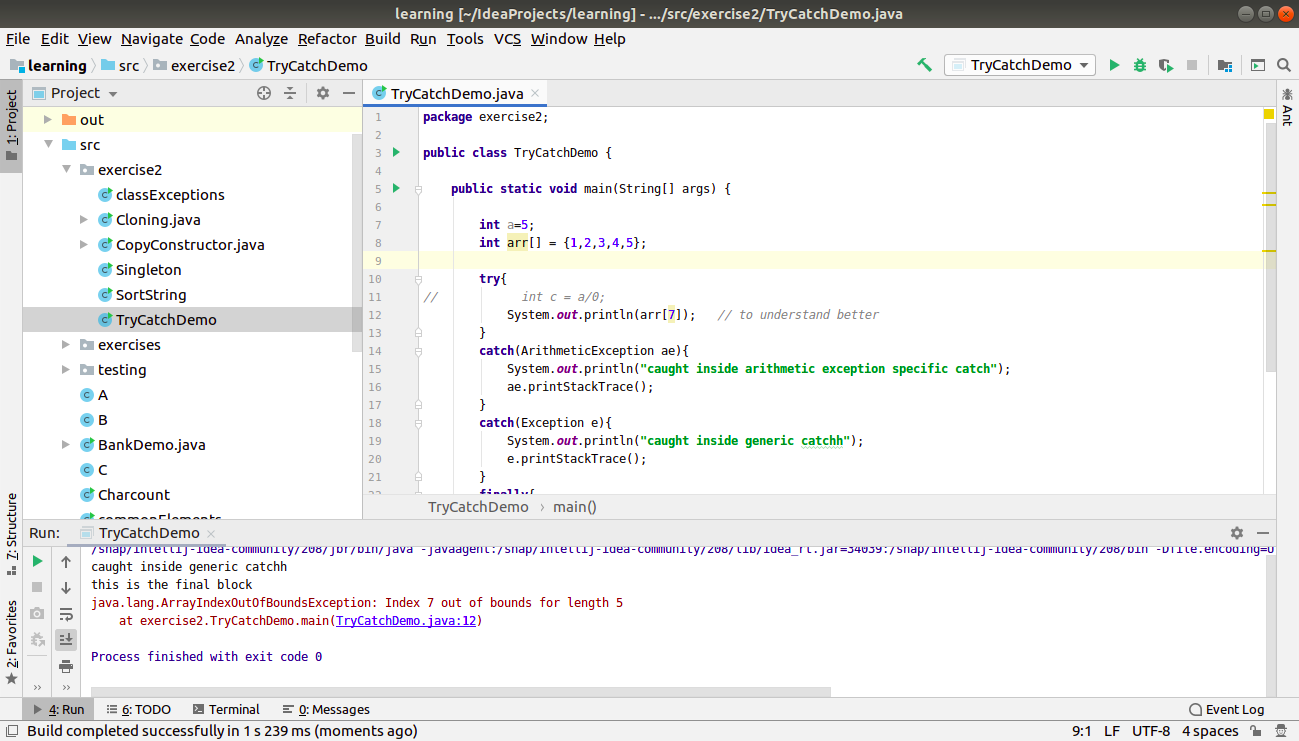
}

}

// in one case - if i run the first statement inside try block



// in another case - if i run the second statement inside try block



**7. WAP to convert seconds into days, hours, minutes and seconds.**

**package** exercise2;

**import** java.util.Scanner;

**public class** DaysConversion {

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

Scanner sc = **new** Scanner(System.***in***);

*// input the seconds value*

**int** temp = sc.nextInt();

*// generate spare seconds [which will be less than 60 always]*

**int** seconds = temp%60;

*// get the spare minutes [less than 60 always]*

temp = temp/60;

**int** minutes = temp%60;

*// get the spare hours*

temp/=60;

**int** hours = temp%24;

*// now only days are left.*

temp/=24;

**int** days = temp;

System.***out***.println(**"the time is: "**);

System.***out***.print(days + **"days "**);

System.***out***.print(hours + **"hours "**);

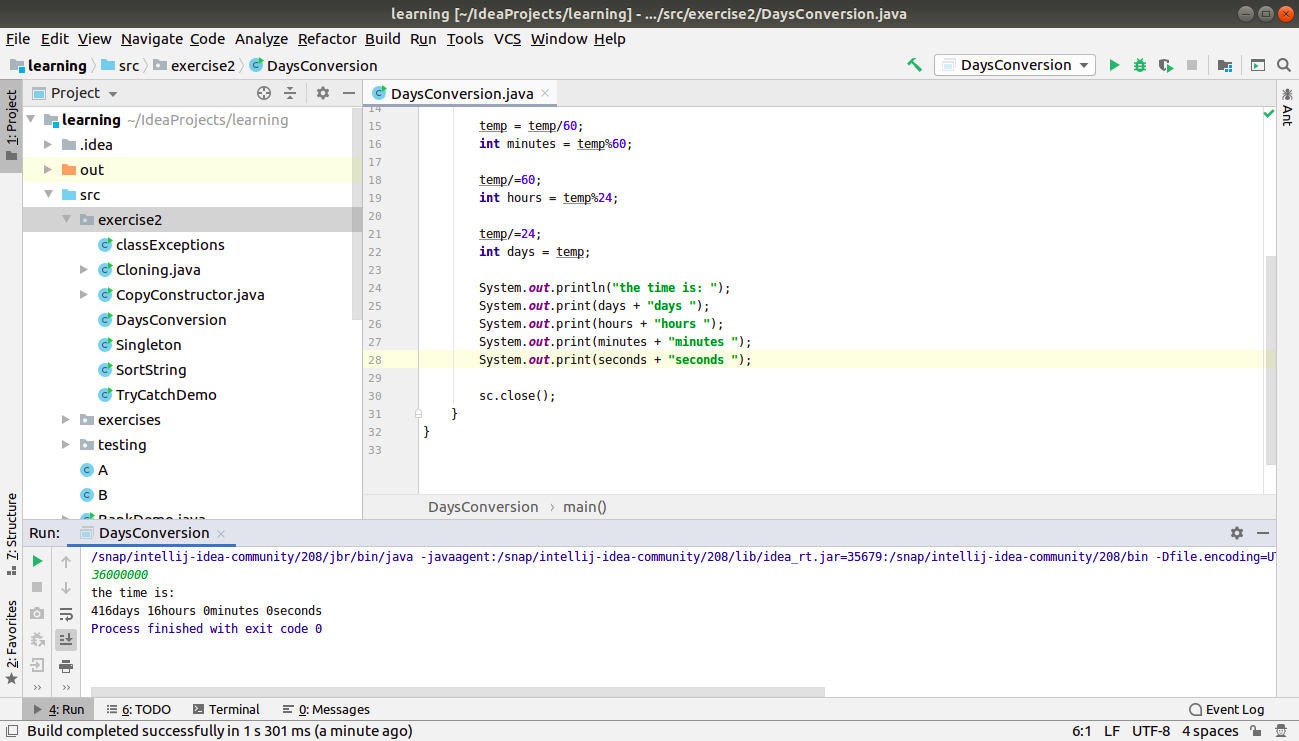
System.***out***.print(minutes + **"minutes "**);

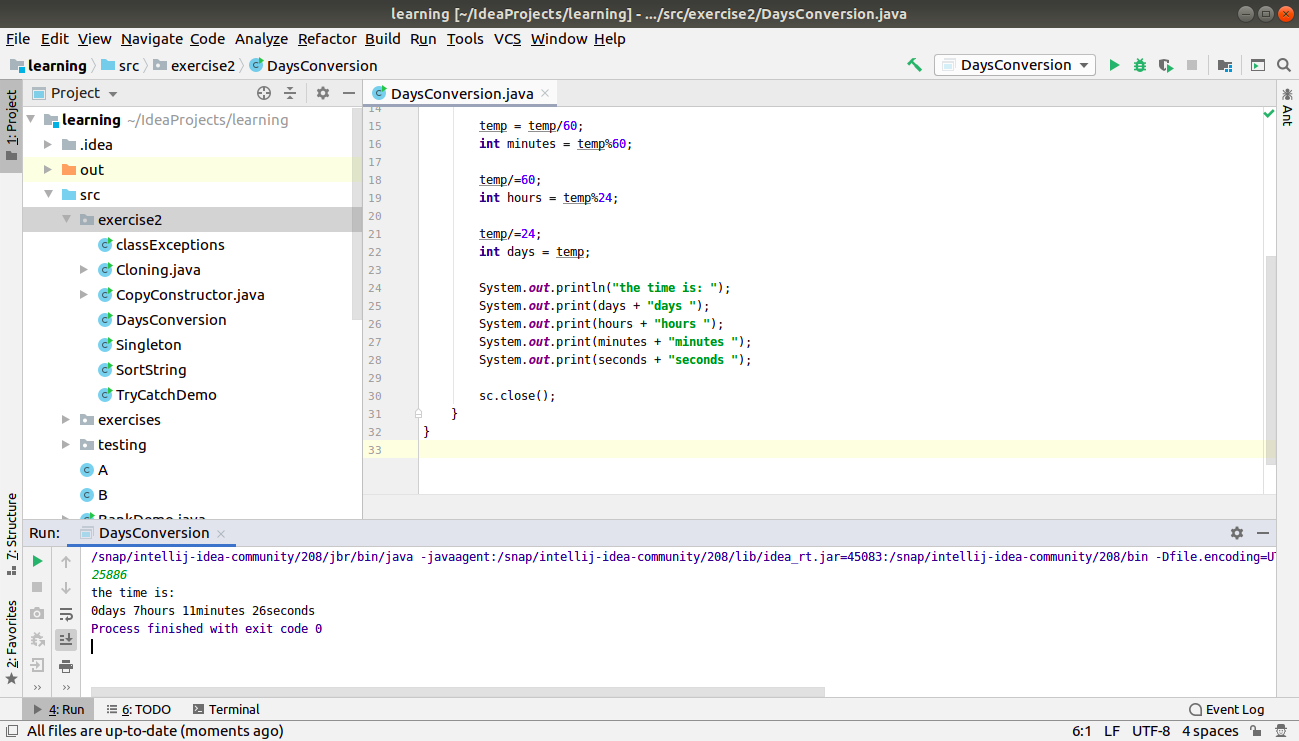
System.***out***.print(seconds + **"seconds "**);

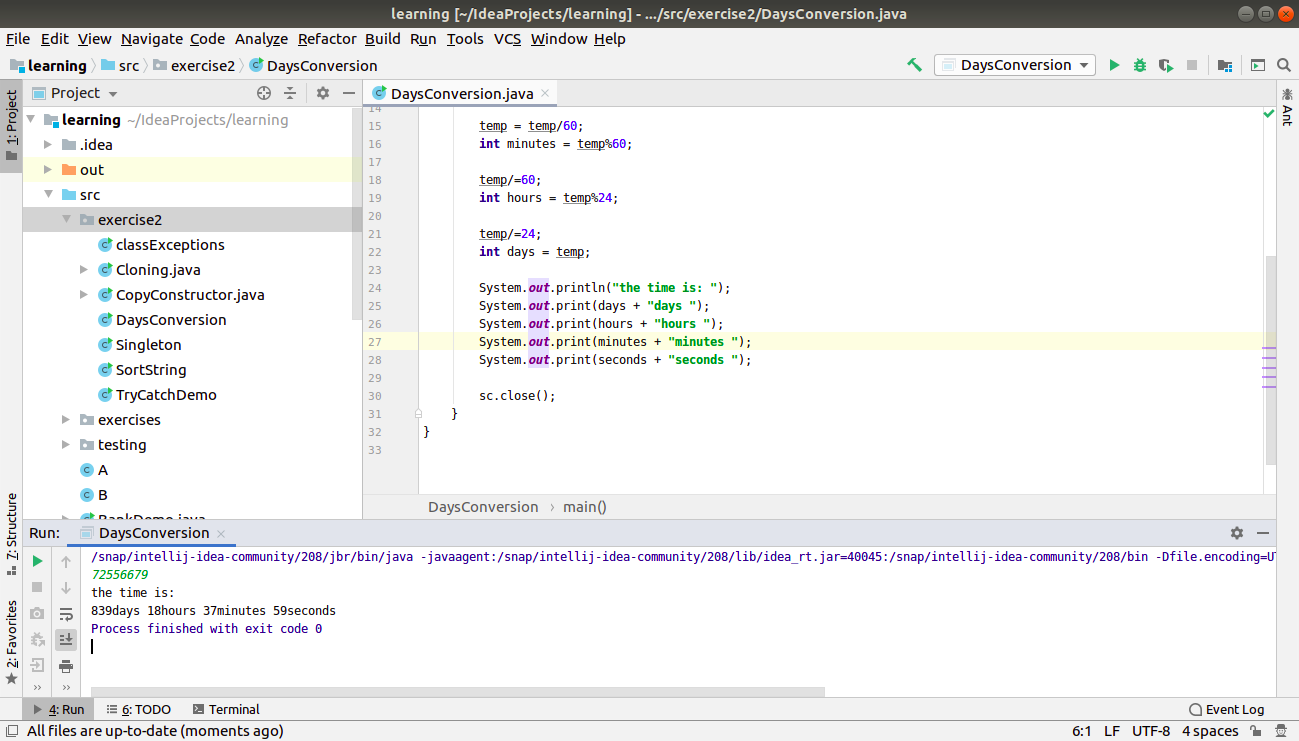
sc.close();

}

}







**8. WAP to read words from the keyboard until the word done is entered. For each word except done, report whether its first character is equal to its last character. For the required loop, use a**

**a)while statement**

**package** exercise2;

**import** java.util.Scanner;

**public class** DoneToken {

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

Scanner sc = **new** Scanner(System.***in***);

**boolean** status;

**while**(sc.hasNext()){

status = **false**;

String word = sc.next();

**if**(word.toLowerCase().equals(**"done"**)){

**break**;

}

**if**(word.charAt(0) == word.charAt(word.length()-1))

status = **true**;

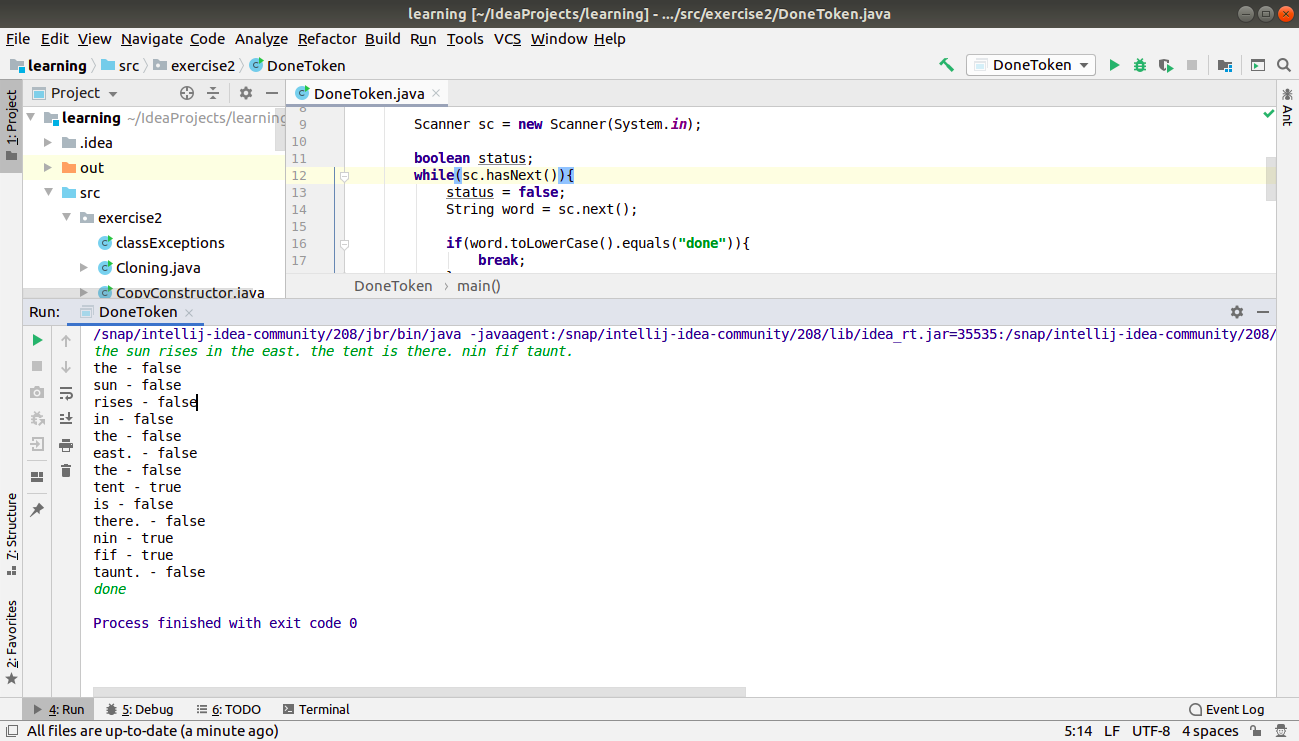
System.***out***.println(word + **" - "** + status);

}

sc.close();

}

}



**b)do-while statement**

**package** exercise2;

**import** java.util.Scanner;

**public class** DoneToken {

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

Scanner sc = **new** Scanner(System.***in***);

**boolean** status;

**do**{

status = **false**;

String word = sc.next();

**if**(word.toLowerCase().equals(**"done"**)){

**break**;

}

**if**(word.charAt(0) == word.charAt(word.length()-1))

status = **true**;

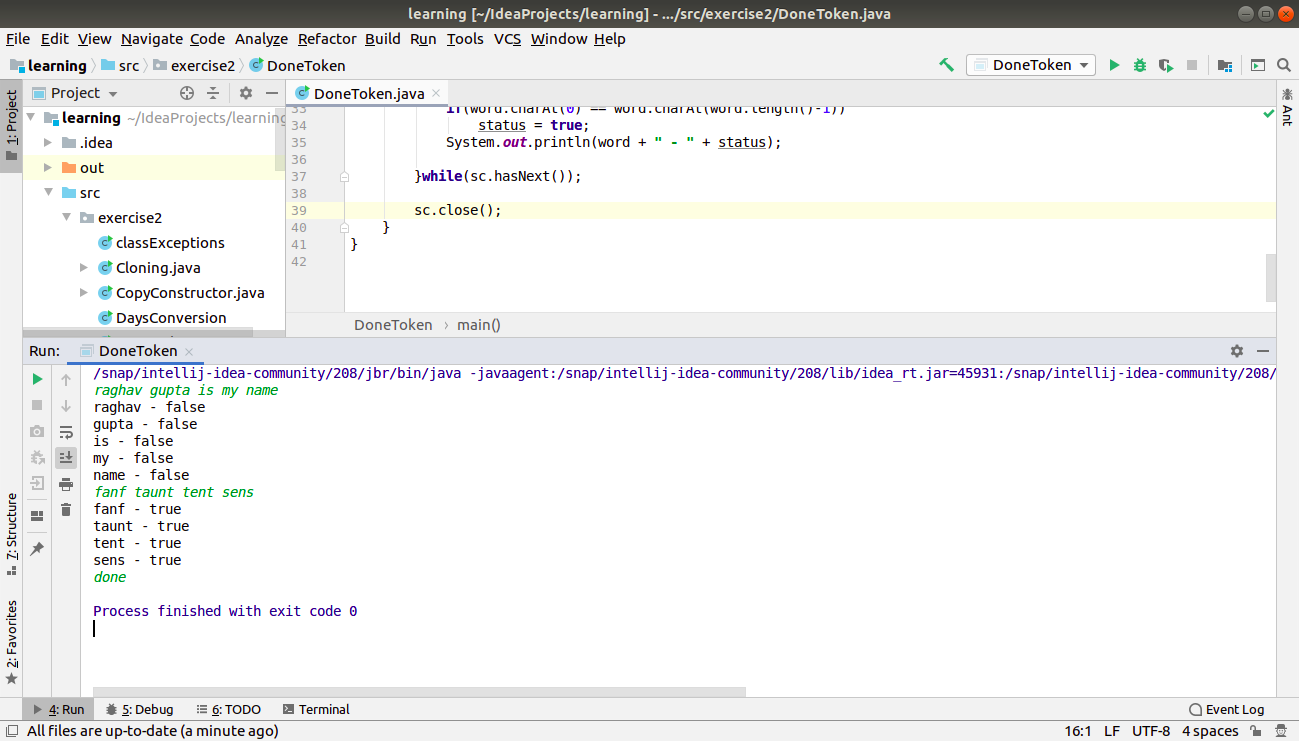
System.***out***.println(word + **" - "** + status);

}**while**(sc.hasNext());

sc.close();

}

}



**9. Design classes having attributes for furniture where there are wooden chairs and tables, metal chairs and tables. There are stress and fire tests for each products.**

**package** exercise2;

**abstract class** Furniture{

**int cost**;

String **brand**;

**int numOfLegs**;

**public abstract void** stressTest();

**public abstract void** fireTest();

**public void** setDetails(**int** c, String b, **int** n){

**cost** = c;

**brand** = b;

**numOfLegs** = n;

}

**public void** showDetails(){

System.***out***.println(**"cost - "** + **cost**);

System.***out***.println(**"brand - "** + **brand**);

System.***out***.println(**"number of legs - "** + **numOfLegs**);

}

}

**abstract class** Chair **extends** Furniture {

**public abstract void** getChairType();

}

**abstract class** Table **extends** Furniture {

**public abstract void** getTableType();

}

**class** MetalChair **extends** Chair{

**public void** stressTest(){

System.***out***.println(**"stress testing performed on metal chair"**);

}

**public void** fireTest(){

System.***out***.println(**"fire test performed on metal chair"**);

}

**public void** getChairType(){

System.***out***.println(**"this is a metal chair"**);

}

@Override

**public void** showDetails() {

**this**.getChairType();

**super**.showDetails();

}

}

**class** WoodenChair **extends** Chair{

**public void** stressTest(){

System.***out***.println(**"stress testing performed on wooden chair"**);

}

**public void** fireTest(){

System.***out***.println(**"fire test performed on wooden chair"**);

}

**public void** getChairType(){

System.***out***.println(**"this is a wooden chair"**);

}

@Override

**public void** showDetails() {

**this**.getChairType();

**super**.showDetails();

}

}

**class** MetalTable **extends** Table{

**public void** stressTest(){

System.***out***.println(**"stress testing performed on metal table"**);

}

**public void** fireTest(){

System.***out***.println(**"fire test performed on metal table"**);

}

**public void** getTableType(){

System.***out***.println(**"this is a metal table"**);

}

@Override

**public void** showDetails() {

**this**.getTableType();

**super**.showDetails();

}

}

**class** WoodenTable **extends** Table{

**public void** stressTest(){

System.***out***.println(**"stress testing performed on wooden table"**);

}

**public void** fireTest(){

System.***out***.println(**"fire test performed on wooden table"**);

}

**public void** getTableType(){

System.***out***.println(**"this is a wooden table"**);

}

@Override

**public void** showDetails() {

**this**.getTableType();

**super**.showDetails();

}

}

**public class** FurnitureDemo {

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

WoodenChair wc = **new** WoodenChair();

wc.setDetails(1500, **"manhatton"**, 4);

wc.showDetails();

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

MetalChair mc = **new** MetalChair();

mc.setDetails(2000, **"godrej"**, 3);

mc.showDetails();

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

WoodenTable wt = **new** WoodenTable();

wt.setDetails(10000, **"manhatton"**, 4);

wt.showDetails();

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

MetalTable mt = **new** MetalTable();

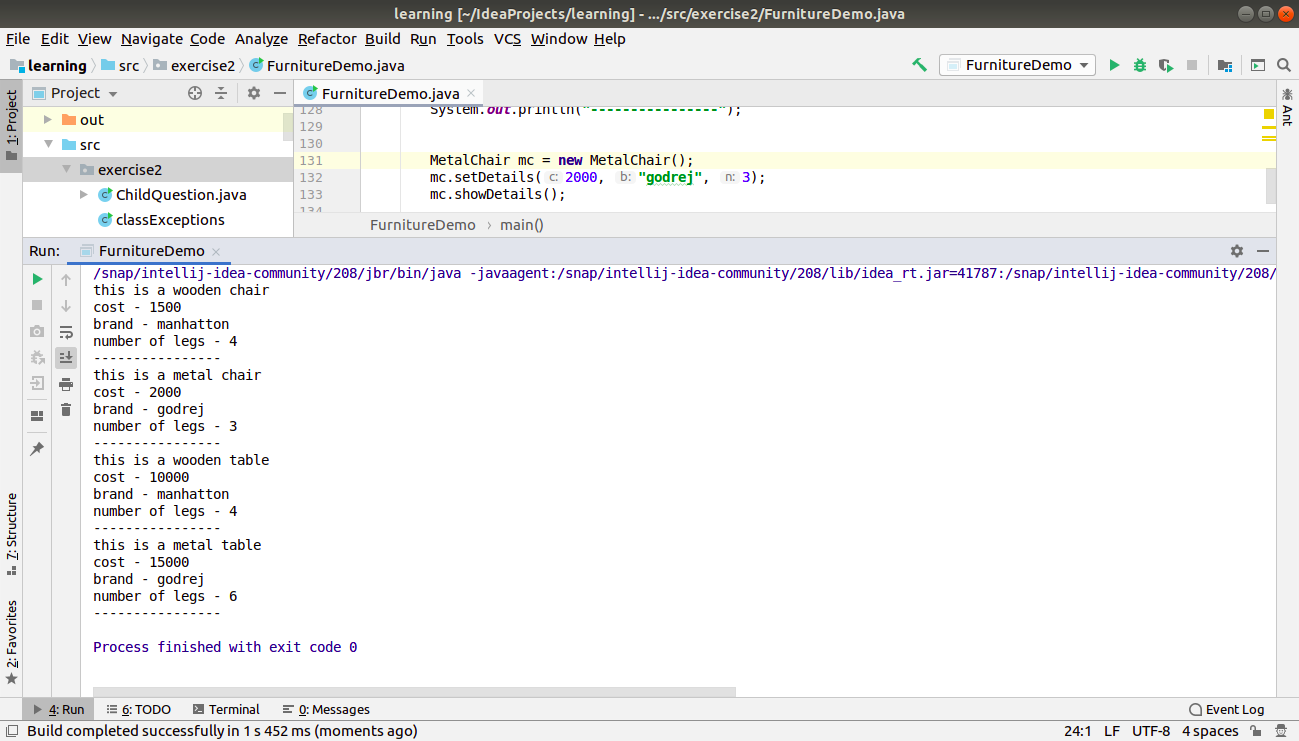
mt.setDetails(15000, **"godrej"**, 6);

mt.showDetails();

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

}

}



**10. Design classes having attributes and method(only skeleton) for a coffee shop. There are three different actors in our scenario and i have listed the different actions they do also below**

**\* Customer**

**- Pays the cash to the cashier and places his order, get a token number back**

**- Waits for the intimation that order for his token is ready**

**- Upon intimation/notification he collects the coffee and enjoys his drink**

**( Assumption: Customer waits till the coffee is done, he won’t timeout and cancel the order. Customer always likes the drink served. Exceptions like he not liking his coffee, he getting wrong coffee are not considered to keep the design simple.)**

**\* Cashier**

**- Takes an order and payment from the customer**

**- Upon payment, creates an order and places it into the order queue**

**- Intimates the customer that he has to wait for his token and gives him his token**

**( Assumption: Token returned to the customer is the order id. Order queue is unlimited. With a simple modification, we can design for a limited queue size)**

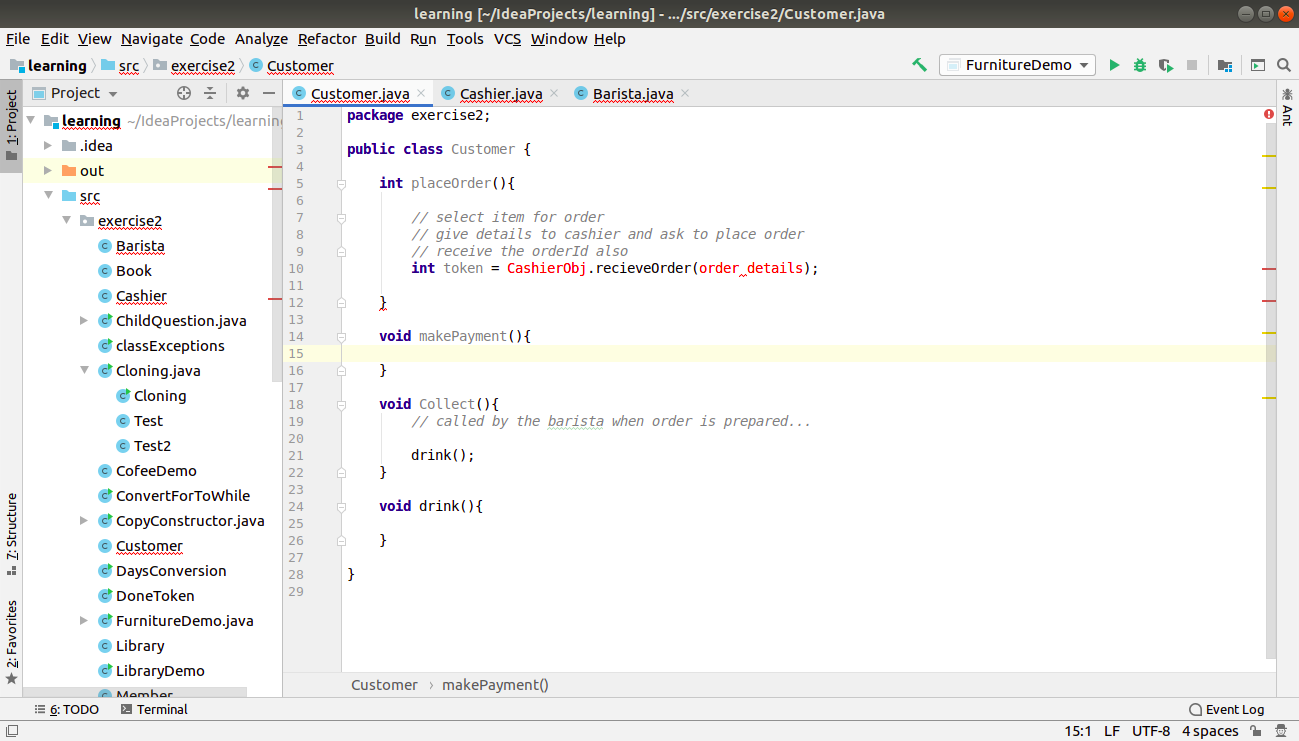
**\* Barista**

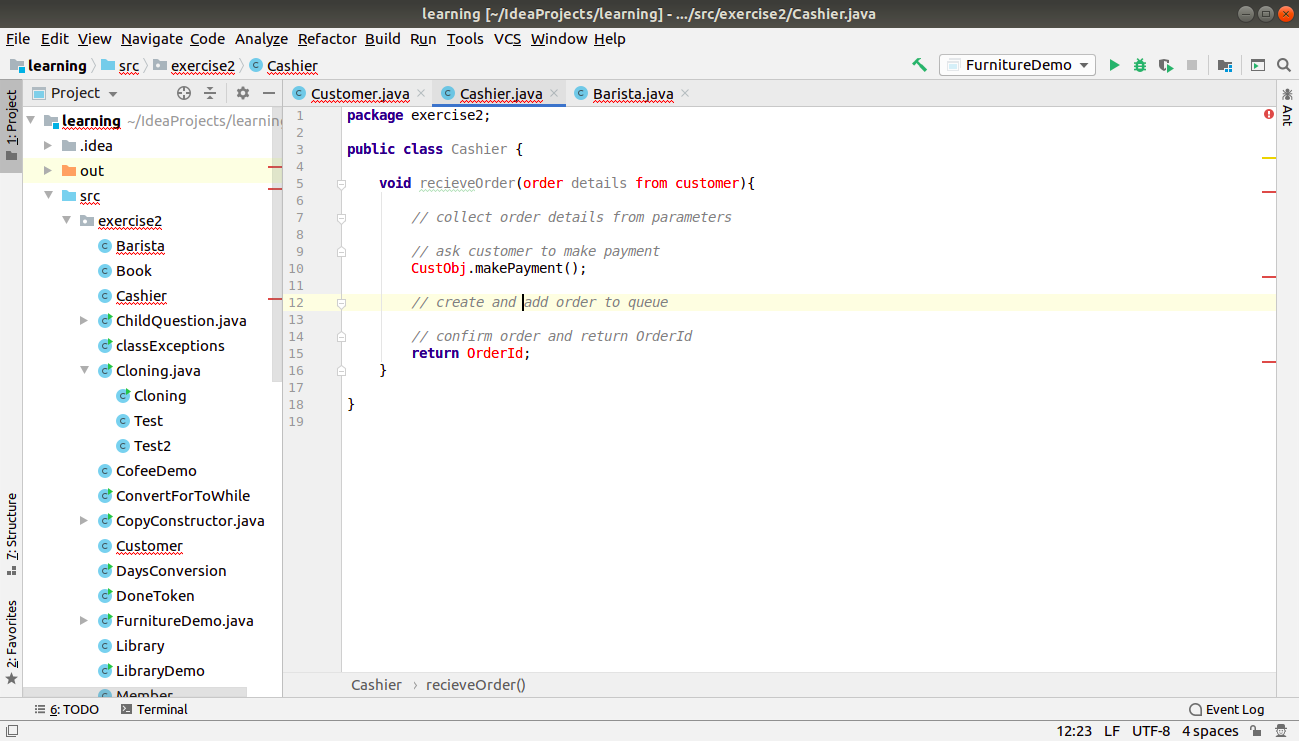
**- Gets the next order from the queue**

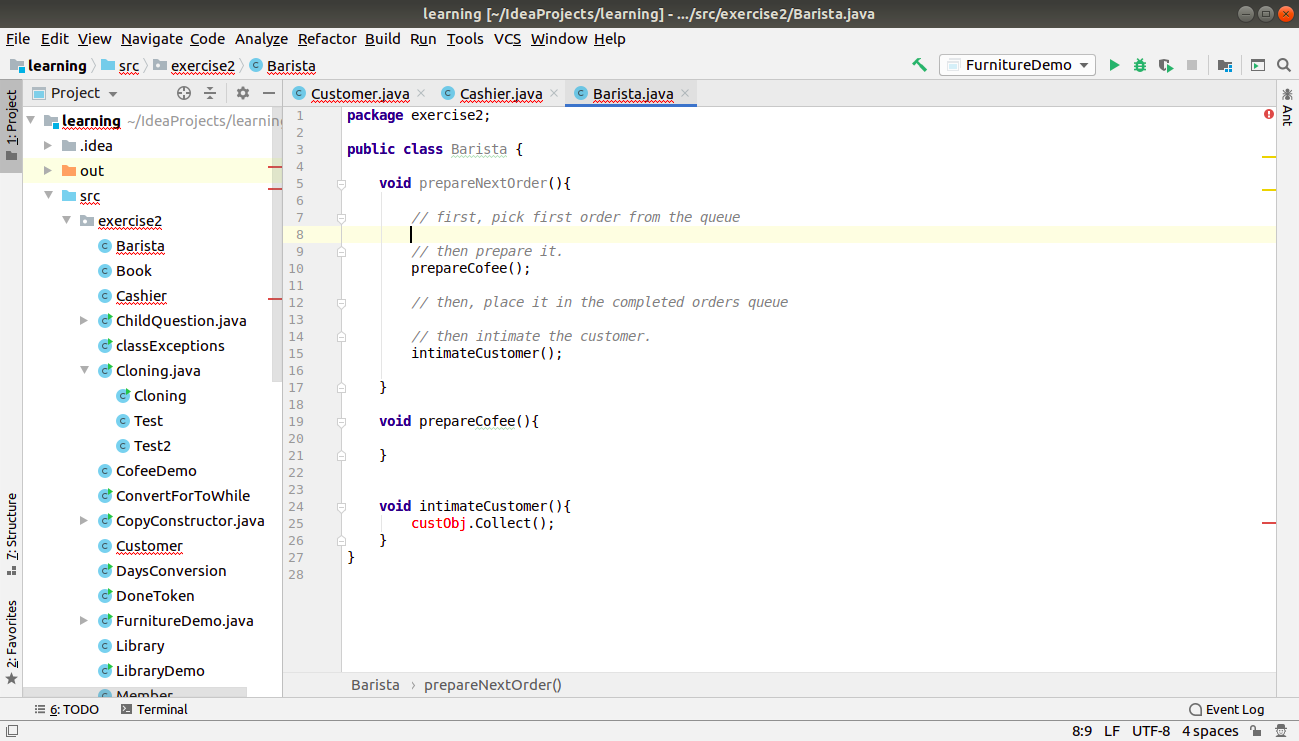
**- Prepares the coffee**

**- Places the coffee in the completed order queue**

**- Places a notification that order for token is ready**

****

****

****

**11. Convert the following code so that it uses nested while statements instead of for statements:**

**int s = 0;**

**int t = 1;**

**for (int i = 0; i < 10; i++)**

**{**

**s = s + i;**

**for (int j = i; j > 0; j−−)**

**{**

**t = t \* (j - i);**

**}**

**s = s \* t;**

**System.out.println("T is " + t);**

**}**

**System.out.println("S is " + s);**

**package** exercise2;

**public class** ConvertForToWhile {

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

**int** s = 0;

**int** t = 1;

**int** i=0;

**while**(i < 10)

{

s = s + i;

**int** j=i;

**while**(j>0)

{

t = t \* (j - i);

j--;

}

s = s \* t;

System.***out***.println(**"T is "** + t);

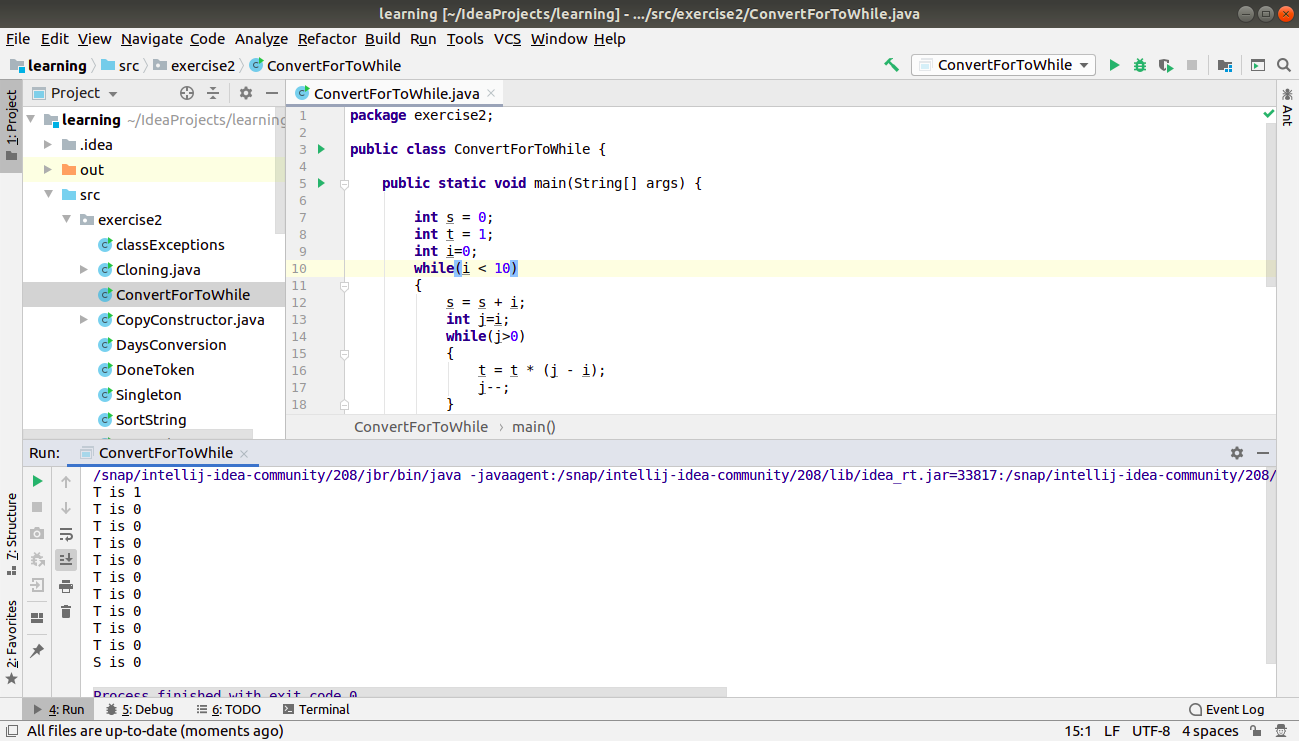
i++;

}

System.***out***.println(**"S is "** + s);

}

}



**12.What will be the output on new Child(); ?**

**class Parent extends Grandparent {**

**{**

**System.out.println("instance - parent");**

**}**

**public Parent() {**

**System.out.println("constructor - parent");**

**}**

**static {**

**System.out.println("static - parent");**

**}**

**}**

**class Grandparent {**

**static {**

**System.out.println("static - grandparent");**

**}**

**{**

**System.out.println("instance - grandparent");**

**}**

**public Grandparent() {**

**System.out.println("constructor - grandparent");**

**}**

**}**

**class Child extends Parent {**

**public Child() {**

**System.out.println("constructor - child");**

**}**

**static {**

**System.out.println("static - child");**

**}**

**{**

**System.out.println("instance - child");**

**}**

**}**

**package** exercise2;

**class** Grandparent {

**static** {

System.***out***.println(**"static - grandparent"**);

}

{

System.***out***.println(**"instance - grandparent"**);

}

**public** Grandparent() {

System.***out***.println(**"constructor - grandparent"**);

}

}

**class** Parent **extends** Grandparent {

{

System.***out***.println(**"instance - parent"**);

}

**public** Parent() {

System.***out***.println(**"constructor - parent"**);

}

**static** {

System.***out***.println(**"static - parent"**);

}

}

**class** Child **extends** Parent {

**public** Child() {

System.***out***.println(**"constructor - child"**);

}

**static** {

System.***out***.println(**"static - child"**);

}

{

System.***out***.println(**"instance - child"**);

}

}

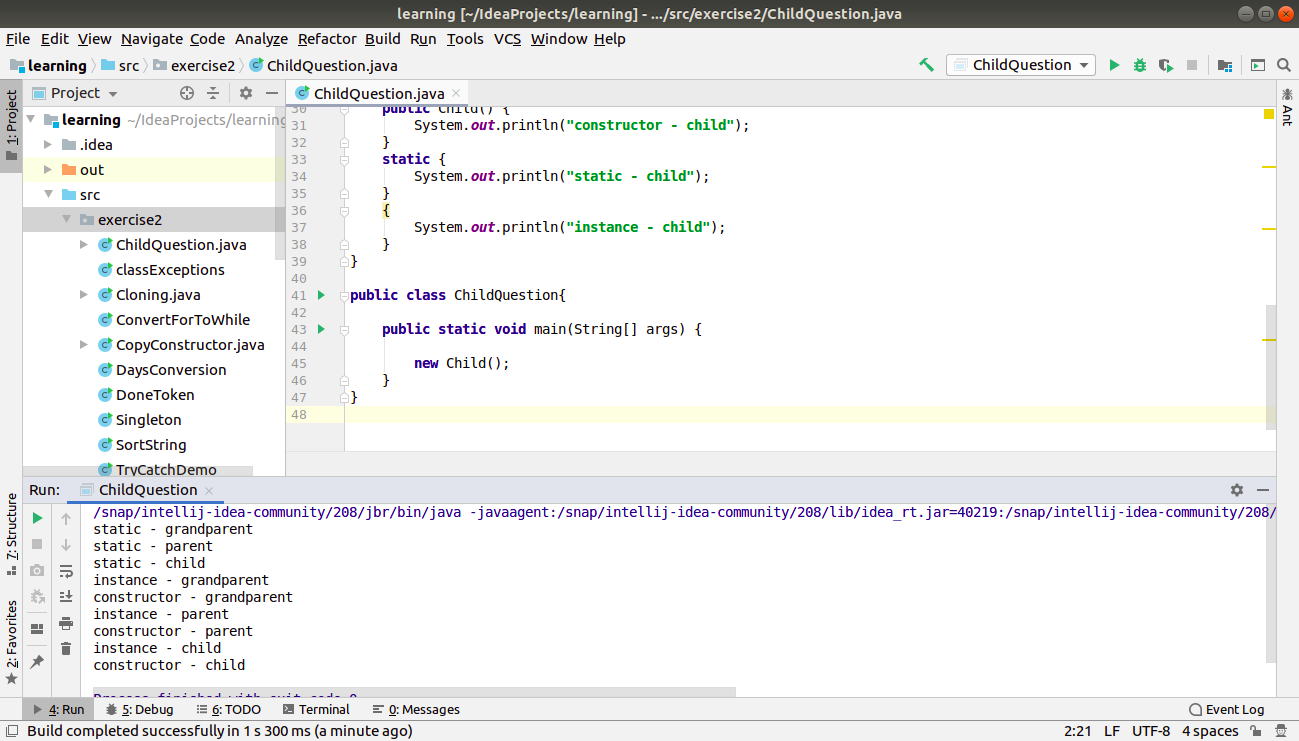
**public class** ChildQuestion{

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

**new** Child();

}

}



**Q13. Create a custom exception that do not have any stack trace.**

**package** exercise2;

**public class** MyException **extends** Exception{

**public** MyException(String s){

**super**(s);

}

**public** Throwable fillInStackTrace(){

**return this**;

}

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

**try**{

**throw new** MyException(**"testing exception"**);

}

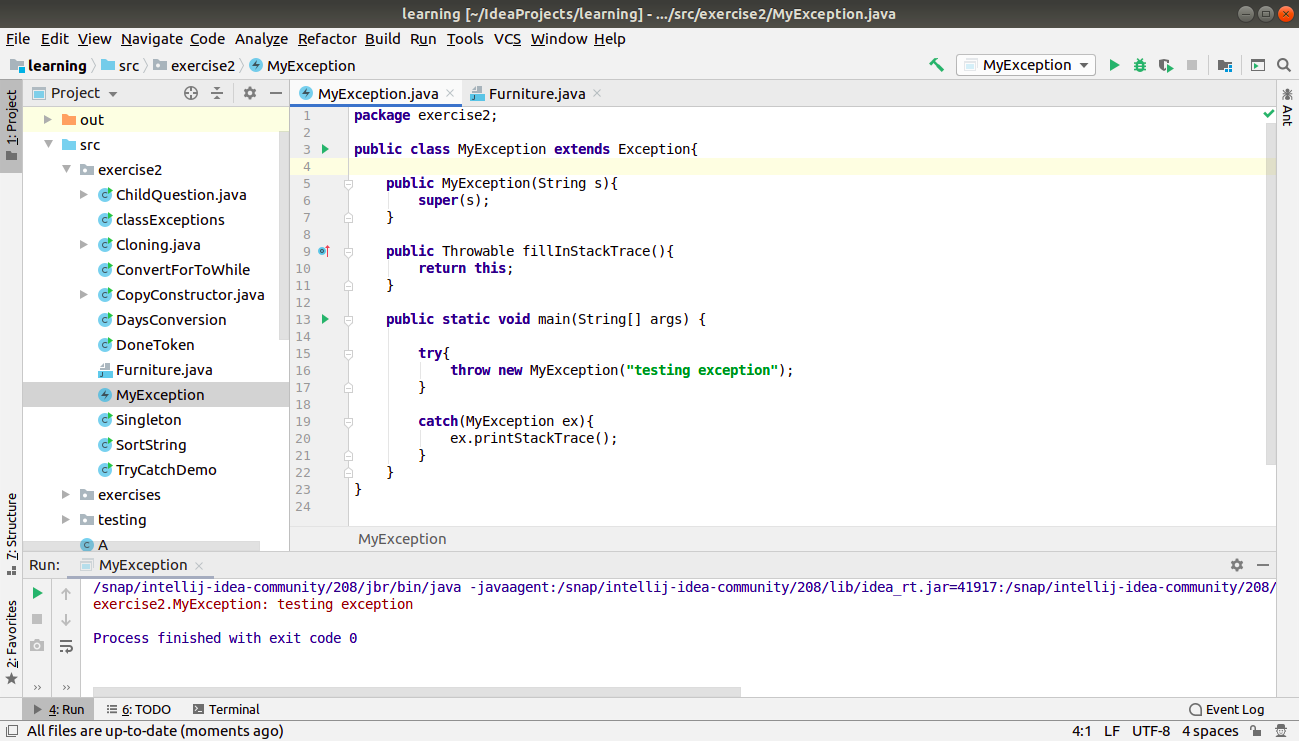
**catch**(MyException ex){

ex.printStackTrace();

}

}

}

****