Java Fullstack

Course Agenda

* Core Java
* JDBC
* Javascript (ES new features)
* React.js
* Angular Framework
* Spring
* Spring Boot
* Spring Microservices
* DevOps and AWS

Core Java

Java is a platform independent programming language & it is object oriented as well.

Platform Independent: You can run java programs on any platforms without altering

Object Oriented: It allows you to create applications with real world entities which are called as objects, object will have 2 things mainly

1. properties (data/varaibles/fields)
2. behaviour (actions/functions/methods).

Software required

* JDK (Java Development Kit) - 1.8
* JRE (Java Runtime Environment) comes along with JDK
* Eclipse IDE (tools to develop java applications)

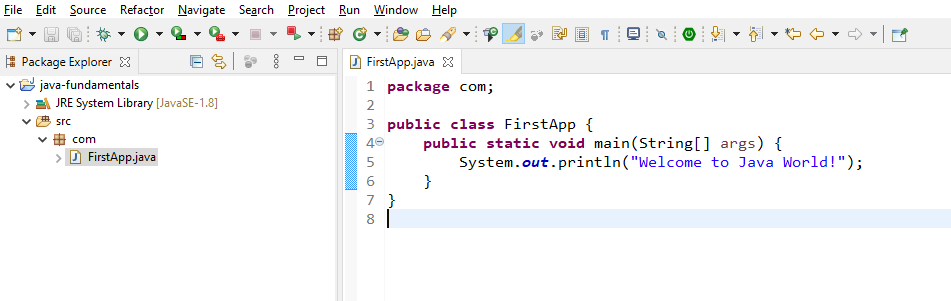
JRE: It is a runtime environment to run java applications, it will have JVM to execute byte codes

Building block of Object Oriented language

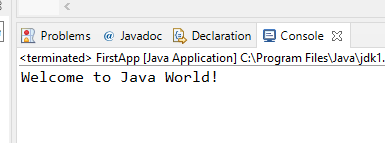
* Class: It is a template of an object, doesn’t take memory in the heap
* Object: It is a real world entity created from the class, takes memory in the heap

How to write our first java program

1. Use Eclipse & Create a java project, if in case you are in JavaEE perspective change it
2. Create a class & name it and also mention the package name
3. Create an entry point method i.e., main method
4. Run the main method.



Output:



Java Features

* Simple to understand, because most of the low level functionalities are abstract
* Object Oriented
* Platform Independent
* Architecture Neutral(x64, X86)
* Distributed applications
* Robust
* Secured
* Large Community
* Open Source

Fundamentals of Java

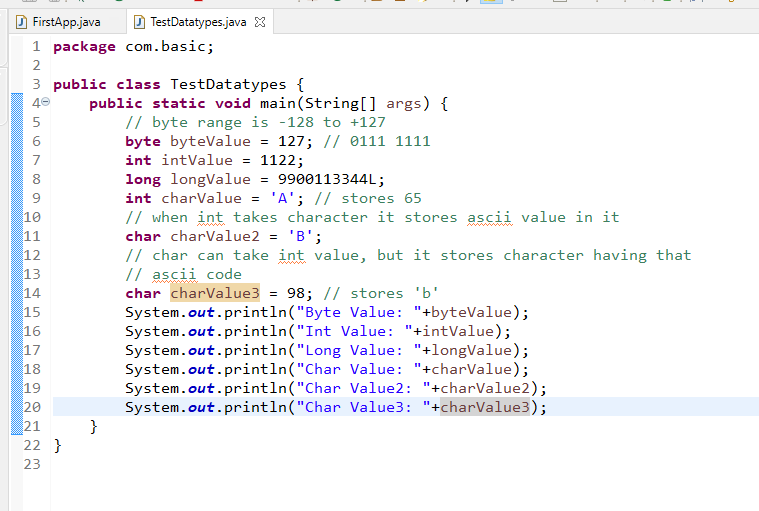
* Datatatypes
* Keywords
* Operators
* Conditional Statements
* Arrays
* Loops
* Variables
* Methods
* Classes
* Objects

Datatypes in Java

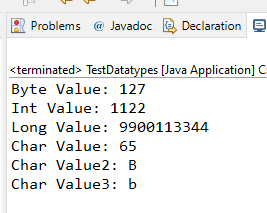
These are set of keywords used to create variables to store some value like numbers, characters, boolean, object and etc.

There are 2 types of datatypes in java

1. Primitive type: These are simple types whose size is defined
   1. byte = 1 byte
   2. short = 2 bytes
   3. int = 4 bytes
   4. long = 8 bytes
   5. float = 4 bytes
   6. double = 8 bytes
   7. char = 2 bytes
   8. boolean = 1 byte
2. Derived type: These are complex types that can store multiple values
   1. array
   2. class
   3. interface
   4. enum



Output:



Keywords in Java

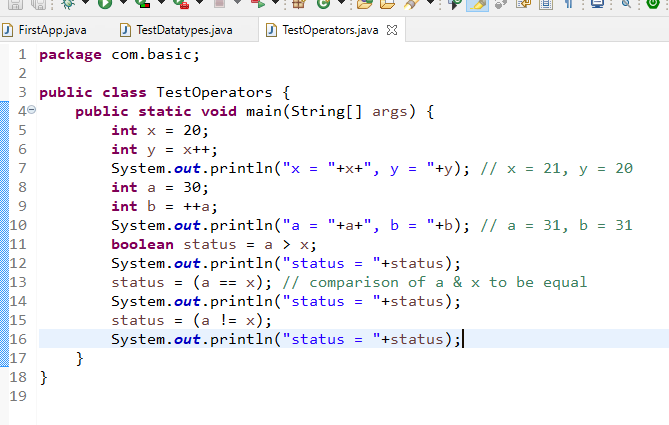
These are reserved words in Java, which must not be used to name the variables, methods or classes, they solve specific purpose

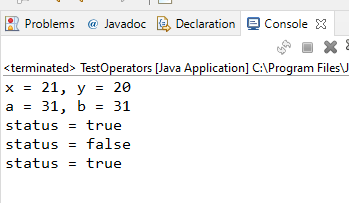
int, char, float, double, boolean, long, final, public, private, class, protected, static, void, return, if, for, do, while, switch, break, continue, abstract, interface, enum, extends, super, this and so on.

Operators in Java

+, -, \*, /, %, ++, --, =, ==, <, >, <=, >=, !=, &&, ||

TestOperators.java

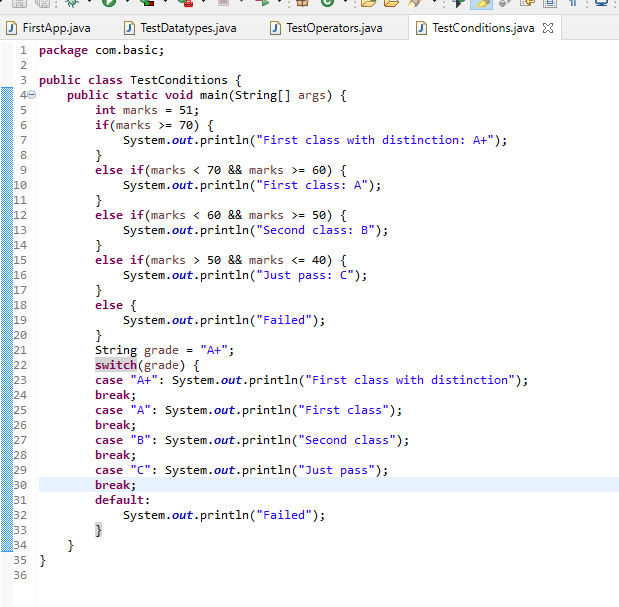




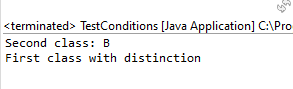
Conditional Statements

These are used when you want to execute the statements based on some conditions, we have following conditional statements

* if
* if else
* if else if .. else
* switch



Output:



Scanner class:

It is a predefined class present in java.util package, which is used to take input from the keyboard, you need to import this class using ‘import java.util.Scanner’

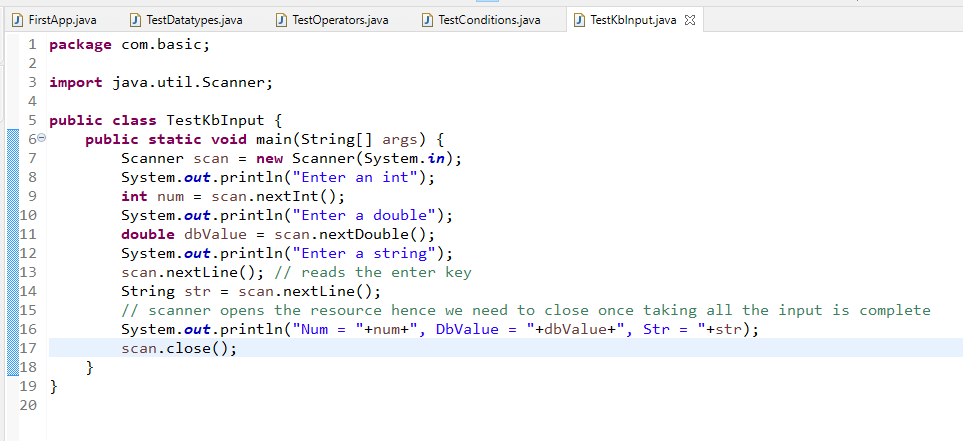
It defines lot of methods to accept different types of value like

nextInt(), nextLong(), next(), nextFloat(), nextDouble() and so on

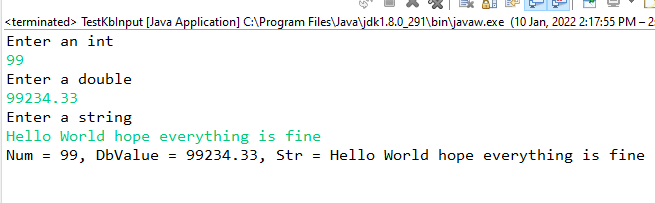
All the above methods you can call by creating scanner object

Scanner scan = new Scanner(System.in);

The above code creates object of scanner, System.in is a reference to the input stream i.e., keyboard



Output:



How to work with arrays

In Java you can create arrays using [], you can create arrays of simple types to complex types

Different ways of initializing array & their values

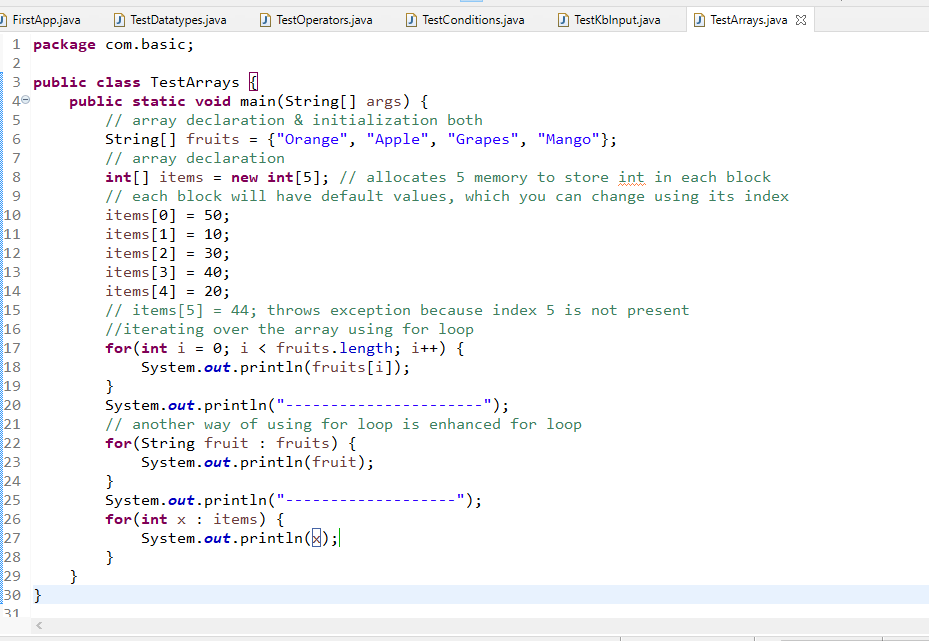
int[] items = {10, 20, 15, 35, 30}; // it is an integer array which is initialized

int[] items = new int[5]; // it is an integer array but not initialized the blocks

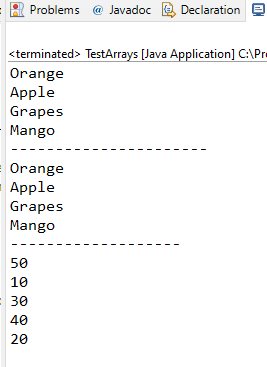
items[0] = 10; items[1] = 30, … items[4] = 50;

String[] fruits = {“Apple”, “Mango”, “Orange”}; // string array which is initialized

You can use loops to iterate over each items of the array i.e., for loop



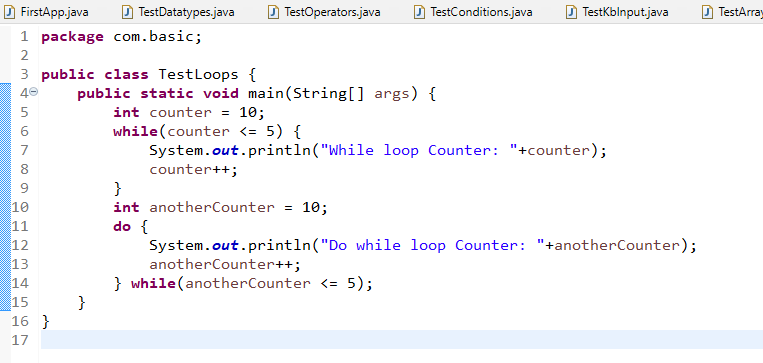
Output:



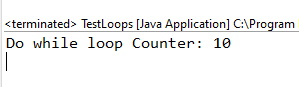
Loops in Java

We have 3 types of loops

1. for: When you want to iterate fixed number of times or you know how many times you want iterate
2. while: When you want to iterate until condition is true
3. do-while: When you want to iterate until condition is true, but in do while the statement is atleast executed once



Output:

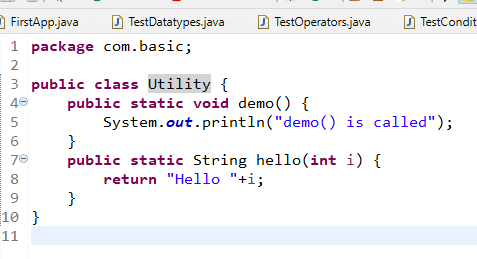


Methods in Java

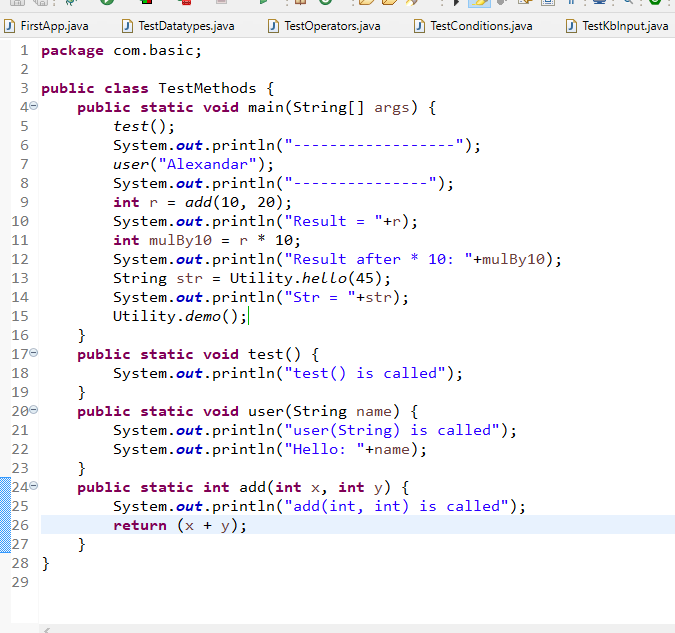
You can create methods in class in 2 ways

* Using static keyword: You can call this method without creating object
* Without static keyword: You can call this method by creating object

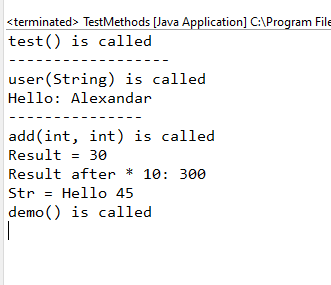
Utility.java



TestMethods.java



Output:



Activities

Create a Exercise folder & day1 sub-folder & keep all the day1 activities here, same way on 2nd day you will create day2 sub-folder and so on.

1. Try all the above examples
2. Create a method that accepts name as a parameter & returns a welcome message, call this method from the main method and display the welcome message

Ex: If you enter Raj the method should return Hello Raj, welcome to java

1. Create a method that accepts 3 numbers as parameters & returns the largest number among them

Ex: If you pass findLargest(3, 9, 1), then it must return 9

1. Create a method to accept 3 numbers & returns the sum of biggest & smallest number.

Ex: If you pass sum(9, 1, 3) then it must return (9 +1), if you pass sum(1, 2, 5) then it must return (5 + 1)

1. Create a method that accepts 3 numbers & prints their digits in words ex: If the input is 754 then the output should be “Seven Five Four”

Day 2 Agenda

* Method Overloading
* Classes & Objects
* OOPs concepts
* Static & Final Keywords
* String, Integer, Float, Double, Object

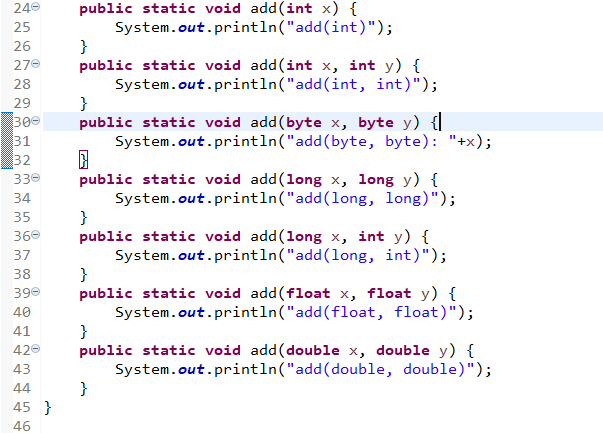
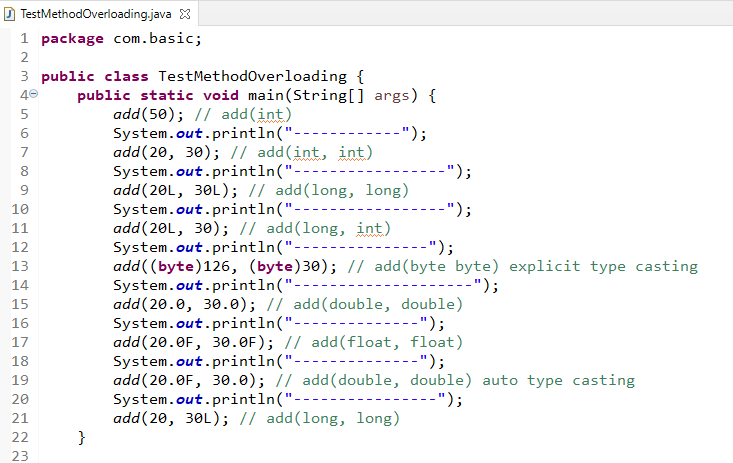
Method Overloading:

Multiple methods with the same name but different parameters or different signature, Compiler can identify which method needs to be called, hence it is also called as compile time polymorphism

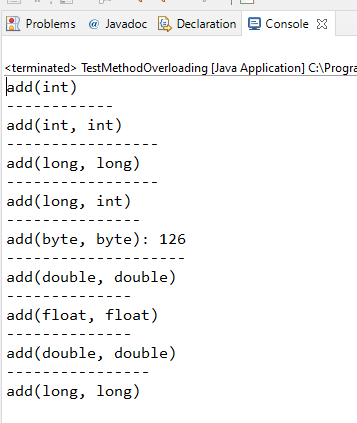
Polymorphism: Poly means many and morphism means forms, overloading is one of the type of polymorphism, there are 2 types of polymorphism

* Method overloading: Compile time polymorphism, methods are identified at compile time itself.
* Method overriding: Runtime polymorphism: Methods are identified at runtime.

TestMethodOverloading.java



Output:

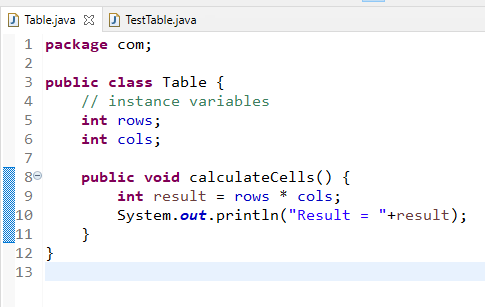


Classes & Objects

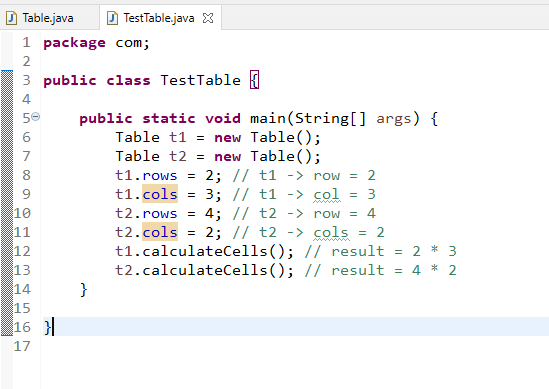
Class is a blueprint of an object, which describes the object

Object is an instance of the class.

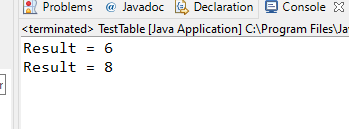
Table.java



TestTable.java



Output:



The above code is initializing the variables of each new object of Table, in a Table class we don’t have any constructor, but compiler creates a default constructor.

Constructor: It is a like a method but name will be same as class name & it should n’t have return types, In Java if you create a class *compiler automatically creates a default constructor if in case user has not created any constructor.*

For ex:

class Test {   
  
}

The compiler creates a default constructor as below

class Test {  
 Test() { …. } // default constructor  
}

Another ex:

class Demo {  
 Demo(int x) { … }  
}

Now the compiler retains the constructor user has provided

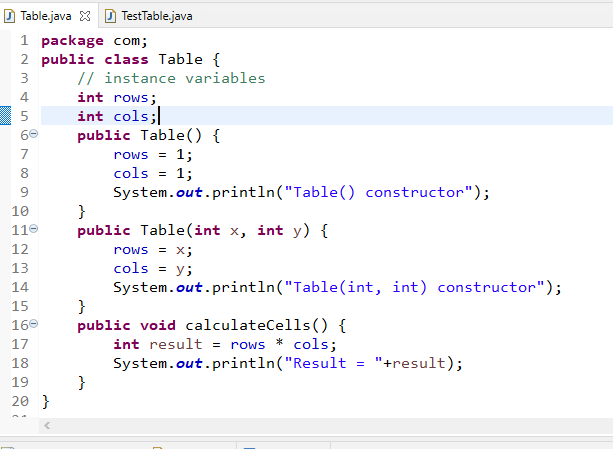
class Demo {  
 Demo(int x) { … }  
}

Another ex;

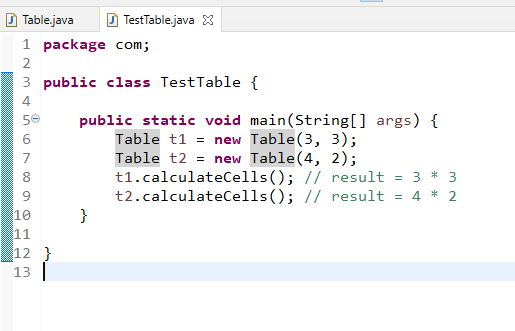
class Hello {  
 Hello() {..}  
 Hello(int x, int y) {…}  
}

Now the compiler retains both the constructors, it means you can also have multiple constructors which means constructor overloading.

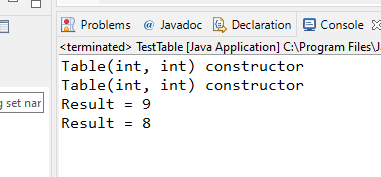
Table.java



TestTable.java



Output:

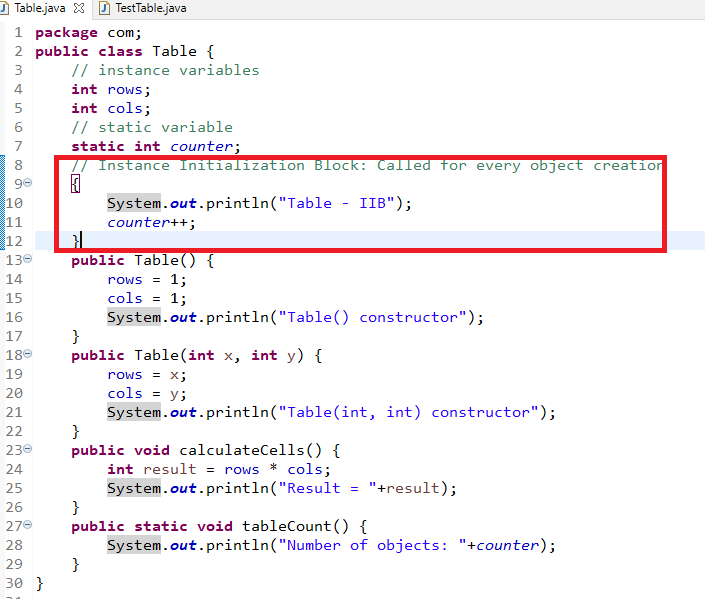


The above code uses parameterized constructor to initialize the instance variables, which also avoids lot of initialization statement at the caller side.

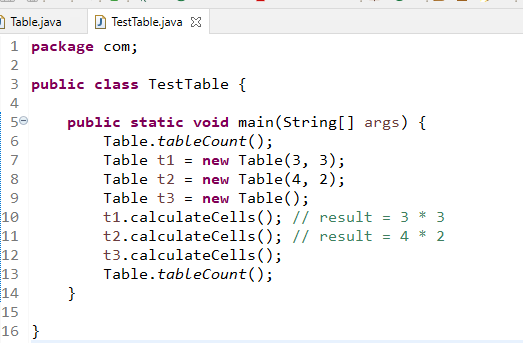
Instance Variables: These are the variables which are created for every object

Static Variables: These are the variables which are shared for all the objects, it will be a single copy created for all the objects of the class, you need to access static variables through class name though you can access it using reference variable, its recommended to access static variables or methods through class name, reason is compiler converts the reference type to class name

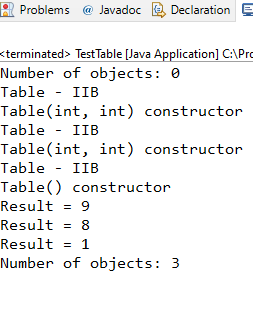
Table.java



TestTable.java



Output:



Features of OOPs

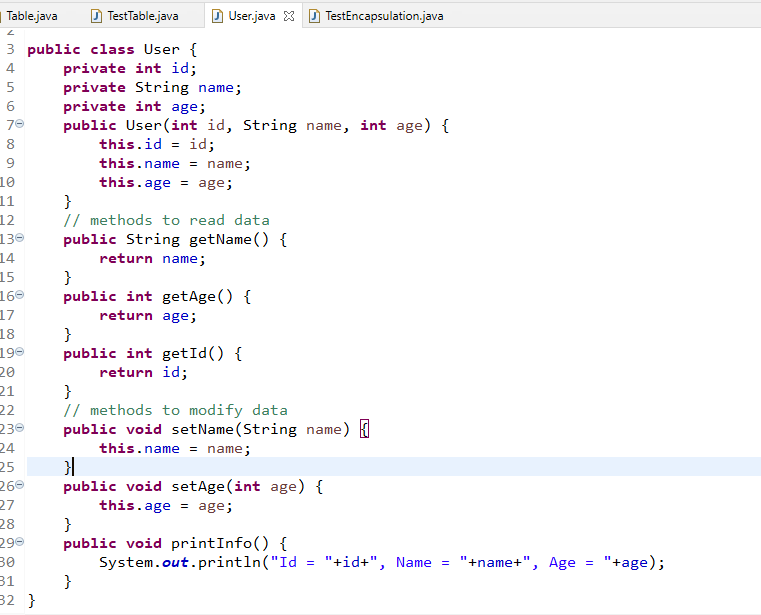
* Encapsulation
* Inheritance
* Polymorphism
* Abstraction

Encapsulation:

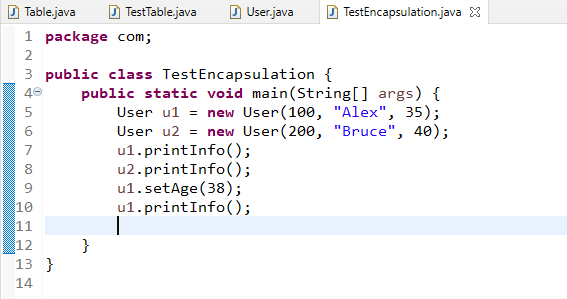
Binding the data & methods in a single entity, you will make data private so that it is accessible only within the class & you will have methods for each variables to read/write which are called getters & setters.

Through encapsulation you will have more control over the data, you can avoid invalid values and also avoid accessing the data if not required.

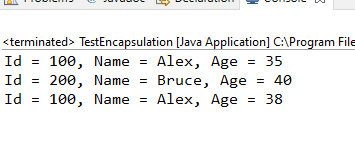
User.java



TestEncapsulation.java



Output:



In Eclipse you have an easier way to generate setters & getters as well as constructor also.

Right Click -> Source -> Generate Setters & Getters

Inheritance

When you want to acquire properties & behaviours of an object from another object you can use inheritance, in java you use extends keyword to achieve inheritance.

class A { }

class B extends A { }

Here A is a super class or parent class whereas B is subclass or child class

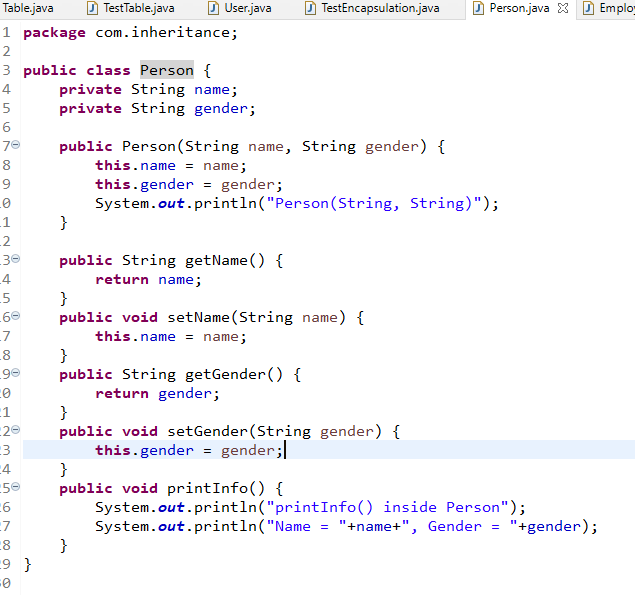
Note: Private members & Constructors wouldn’t be inherited

Note: You can use super keyword to access super class members & constructors

super.member // it accesses super class members from sub class

In Inheritance every constructor of the subclass calls the parent class default constructor from its first line implicitly, you can explicitly call the parameterized constructor of the parent class using super(args) on the first line of the subclass constructor

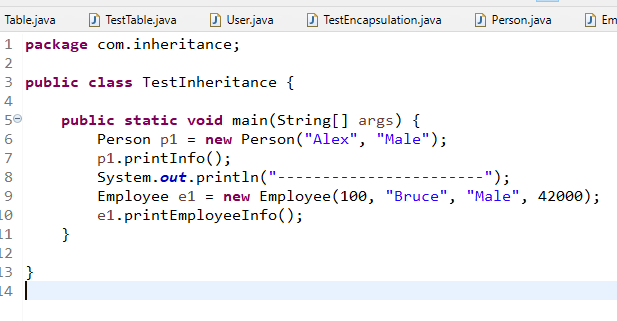
Person.java



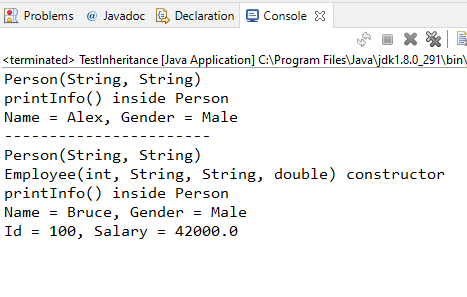
Employee.java



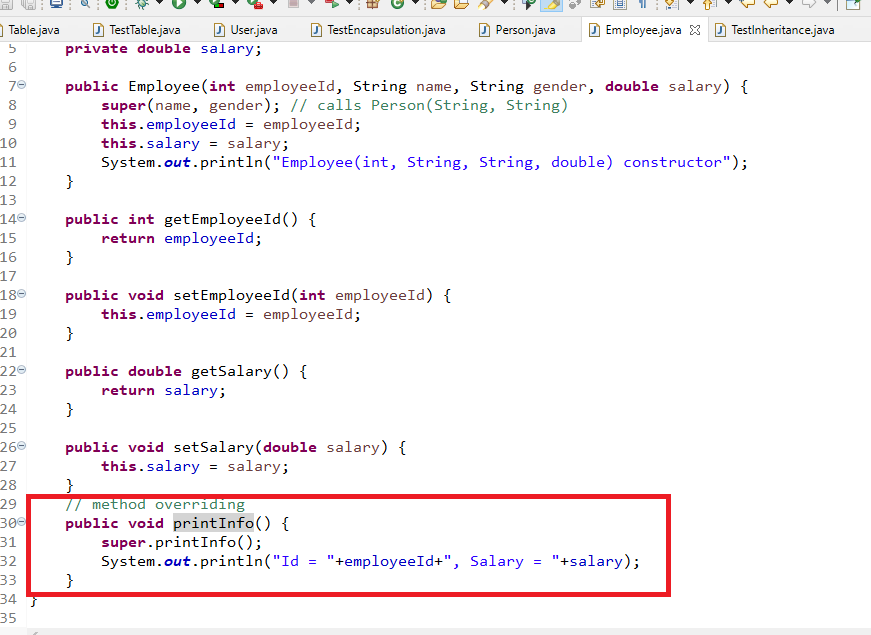
TestInheritance.java



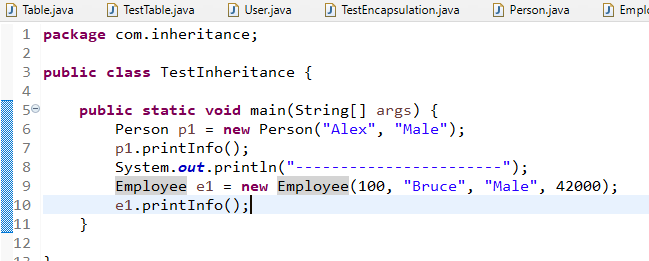
Output:



Instead of creating different methods in subclass to print the data you can override printInfo



Now if you call printInfo on employee object it calls the printInfo present in the Employee object.

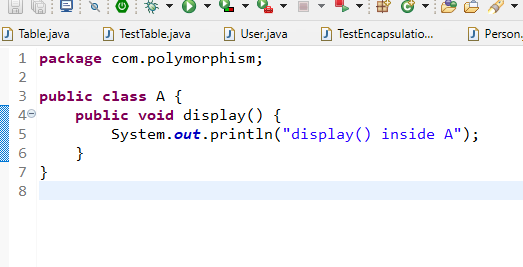


Polymorphism

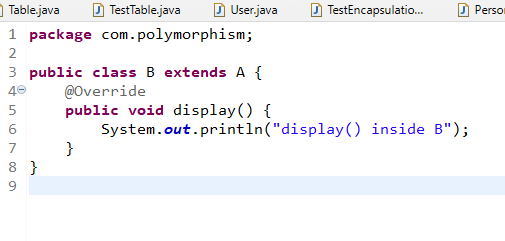
Ability of a method to give multiple results

ex: Single Power button acts like On and Off both, similarly single method can give different results based on the object you are using.

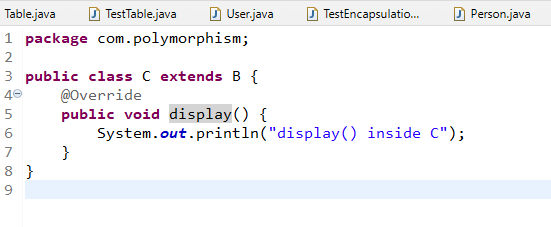
A.java



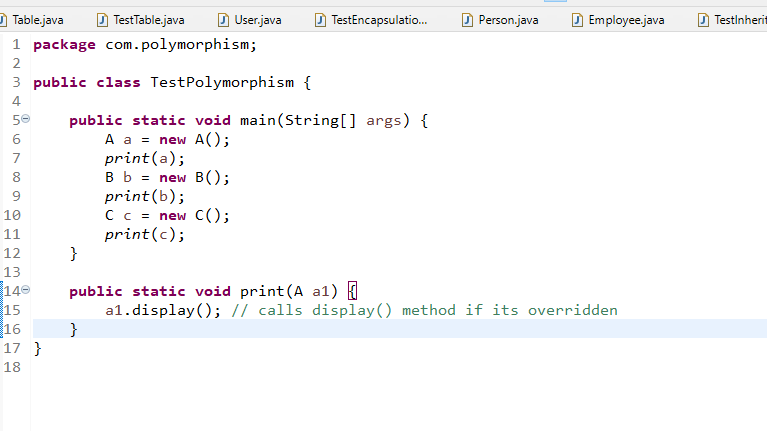
B.java



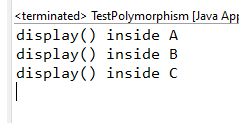
C.java



TestPolymorphism.java



Output:



Activity:

1. For the above inheritance & polymorphism example create display method in Person & override in Employee to print employee information
2. For the same example create a Student & Customer class that should extend Person,
   1. Student class must have rollNo & grade properties & Override display method to print rollNo, name, gender & grade
   2. Customer class must have customerId, accountNumber, balance & Override display method to print customerId, name, gender, accountNumber & balance
   3. From main method create Object of Student, Customer, Employee & Person and pass these objects to a single static print method that all the subclass of person like Customer, Employee, Student and Person also

ie., print(employeeObject), print(studentObject) and so on, it must call display method to print employee, student, customer informations.

1. Define a class Student with the following attributes

* studentId of type integer
* studentName of type String
* city of type String
* marks1 of type integer
* marks2 of type integer
* marks3 of type integer
* feePerMonth of type float
* isEligibleForScholarship of type boolean

Implement the following methods in addition to the setter and getter methods for the various attributes

* getAnualFee() which returns the product of feePerMonth and 12
* getTotalmarks() which returns the sum of marks1, marks2 and marks3
* getAverage() which returns the average of marks1, marks2 and marks3
* getResult() which returns “pass” if the person has scored more than 60 in each subject, or returns “fail” otherwise

Create another class TestMain with the main() method which performs the following actions

* Creates three Student objects
* Populates the objects using the setter methods
* Displays the name of the Student who has the highest total marks
* Prints the name and fee of the Student who pays the least monthly fee
* Prints the name, total marks , average marks , result, and “Scholarship available” or “Scholarship not available” based on the student’s eligibility for every student.

Abstraction

It is a process of hiding the complexity form the user and showing only the necessary details which the user can use without knowing the complexity.

In Java you can achieve abstraction in 2 ways

1. Interface: Fully abstraction
2. Abstract class: Partial abstraction

Interface: It is a kind of class where you will have only abstract methods and constants

abstract methods: These are methods without logics, it will have only method declaration

Interface methods needs to be implemented by classes using implements keyword

Ex:

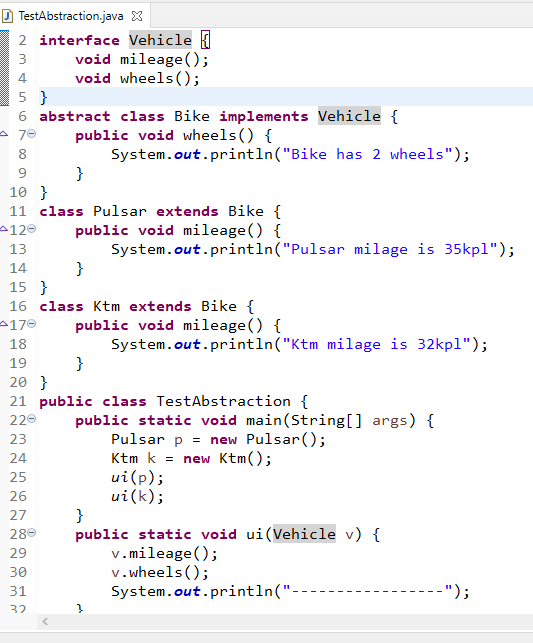
interface A {   
 int x = 10; // by default variables are final & static  
 void test1(); // by default methods are abstract  
 void test2(); // by default methods are abstract  
}

Note: All the members in the interface take public keyword

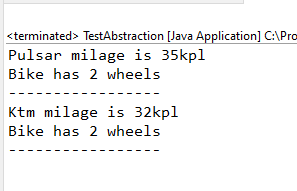
class X implements A {   
 // Now X must implement test1 & test2 mandatorily  
}

abstract class Y implements A {   
 test1() { …… }  
}

TestAbstraction.java



Output:



Activity:

Create an abstract class Car and implements wheels method of Vehicle, then create two classes that extends Car

* Swift >> implement milage
* Ferrari >> implement milage

Pass Switch & Ferrari object to the ui method that takes these objects and calls mileage() & wheels()

Predefined packages & classes & interfaces

In java there are lot of packages that you can import to use their classes, in that java.lang classes can be used without importing.

java.lang.\*: String, Object, StringBuffer, Integer, Long, Exception, ArithmeticException, RuntimeException, Thread, Comparable, Runnable, System

java.io.\*: FileInputStream, FileOutputStream, FileReader, FileWriter, BufferedReader, BufferedWriter, Serializable

java.util.\*: Collection, List, Set, Map, ArrayList, LinkedList, TreeMap, HashSet, LinkedHashMap, LinkedHashMap, Comparator and so on.

java.sql.\*: Connection, Statement, PreparedStatement, ResultSet and so on

All the above classes are public which has many methods that can be used to perform that task, every class will have constructors, methods

Every class has methods that can do the task particular to the class instance, ex: String class will have methods to work on strings,

Integer class has methods to work on int

Thread class has methods to work on threads

Exception class has methods to handle exceptions

FileReader class has methods to perform read operations

Object:

It is a root class for all the classes, by default every class is a subclass of Object

class A { }

class B extends A { }

Here A is the super class of B, but A is a subclass of Object, if a class doesn’t extend any class then by default they extend Object class.

Object class has some common methods that is available to every class, Since every object must have some common behaviours to act like an object, java by default extends Object class,

Some of the methods you can override in every class like

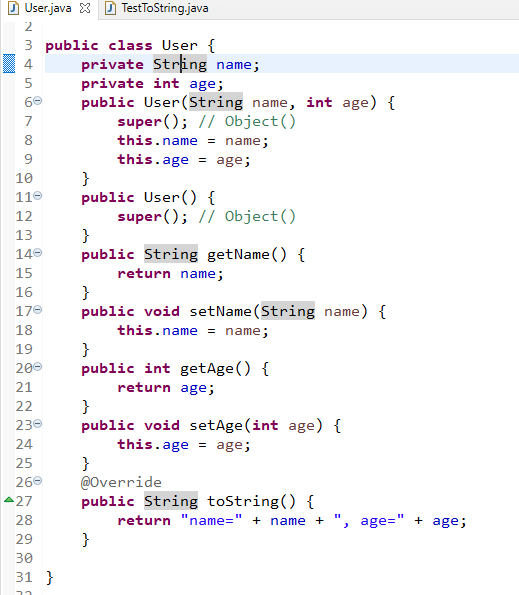
toString(): It returns object representation when you print the object, by default it returns memory address, but you can override to return specific properties

hashCode(): it returns an unique number for every object, it is mainly used in Collection framework, you can override hashCode() if you want to return some property value

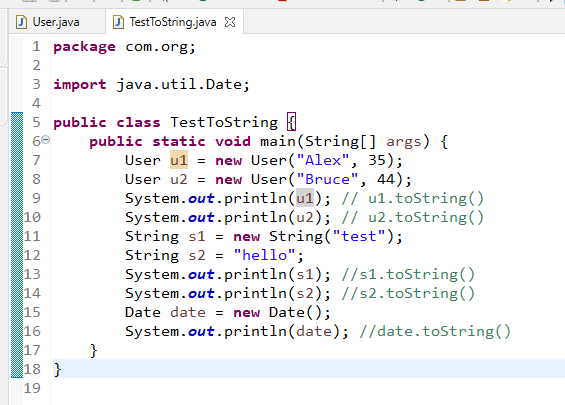
equals(): It returns true when 2 objects are same else it returns false, by default it compares 2 objects address, however you can override equals to compare specific properties or contents of 2 objects to check the equality, it is mainly useful in Collection framework.

Note: When any class overrides hashCode it must override equals also, and if you override equals you must override hashCode as well, i.e., the class must override both to follow certain equals & hashCode rules when it comes to maintaining the object in Set API.

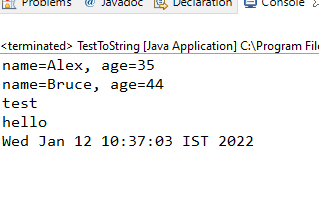
User.java



TestToString.java



Output:



Strings:

There are two types of strings you can create

* Immutable string object: Once the content is created, it can’t be modified
* Mutable string object: The content can be modified after it created

String class: It is used to create immutable string object, once the string is created its content can’t be modified

String methods: toUpperCase(), toLowerCase(), equals(), equalsIgnoreCase(), charAt(), length(), concat(),

Note: In String toString(), equals() & hashCode() of Object class is overridden

StringBuffer class: it is used to create mutable string object, you can modify the string present in string buffer,

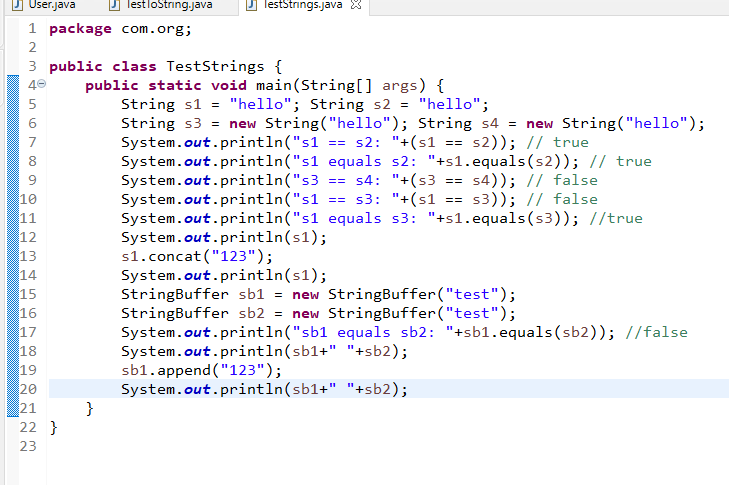
Note: In StringBuffer equals() method & hashCode() is not overridden to compare content, it uses equals() method of Object class only, it has overridden toString() method of Object class

StringBuffer sb1 = new StringBuffer(“test”);  
StringBuffer sb2 = new StringBuffer(“test”);

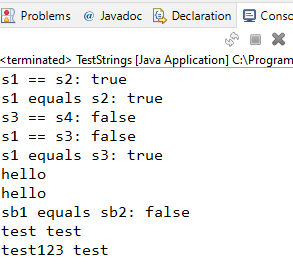
sb1.append(“123”); // now sb1 will be test123

sb2.append(“123”); // now sb2 will be test123

sb1.equals(sb2); // false, equals() method doesn’t compare content



Output:



Activities

Note: All classes should be organized into packages that follow a well-defined hierarchy.

1. Create a class Movie with attributes - movie name, produced by, directed by, duration, year, category (comedy/action/..). Write necessary methods that accept and display the information. Create the constructors based on the below rules.

* Movie name and produced by are mandatory fields and should be supplied at the time of creating the object
* Compiler should raise an error when you try to create Movie object without passing any parameters.
* Write a constructor which accepts all the attributes as parameters while creating the object. From this constructor call the constructor (mentioned at point a) to initialize mandatory fields

1. In the above class create a static variable moviesCount. Write necessary methods to get the values. Every time an object of Movie is created, increment the value of moviesCount variable.
2. Create a movieId field. Make this variable as a readOnly variable (i.e. make it private and write only a getter method). Generate movieId value by using the below formula  
   movieId=”movieName”+”\_”+moviesCount  
   eg. if the Movie name is “Hello” and the value of moviesCount variable is 31, then store Hello\_31
3. Define a new class SpecialMovie which contains all the attributes of Movie and other attributes to store the technology used for soundEffects and visualEffects. Define another class InternationalMovie which contains all attributes of Movie class and other attributes to store country and language.

* Write necessary classes, constructors and methods for storing and displaying additional information. Hint: use super keyword to call the methods/constructor of super class.

1. In the Movie class write a method called showDetails() which concatenates the value of all the attributes and returns it as a String
2. Complete the below tasks using appropriate example of your choice
3. Create Interfaces
4. Create inheritance by extending other interfaces
5. Create class by extending another class and implement more than 1 interface
6. Create a reference variable of an interface.
7. Create a class implementing above interface.
8. Store the object of class created in step e in the reference variable created in step d.
9. Call the methods by using interface reference variable

Exception Handling:

Exceptions are nothing but Runtime errors, which can abruptly stop program execution, hence you must handle the exceptions.

There are 5 keywords in exception handling mechanism

* try
* catch
* finally
* throw
* throws

try block: You write logics that can cause exception

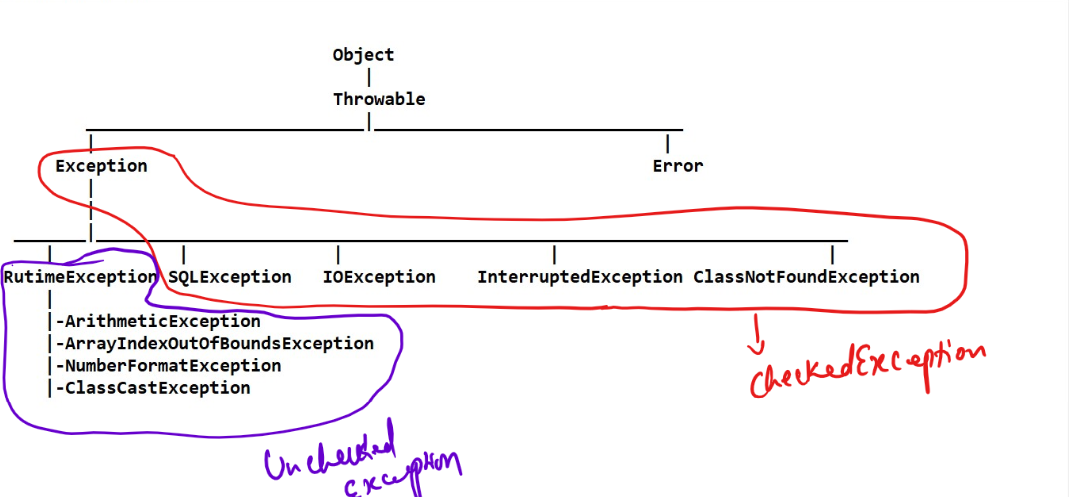
catch block: you can handle the exception thrown by try block, you can have more than one catch block

finally block: It is definitely executed regardless of exception handled or not

throw: manually create the exception like custom exception

throws: it is to propagate the exception to the caller instead of handling it

Exception hierarchy



RuntimeException & their subclasses are unchecked exception, compiler doesn’t force you to handle

Exception & their subclasses except RuntimException are checked exceptions, compiler forces you to handle.

Custom Exception:

You can extend any of the exception class to create your own exception, but if you want to create a checked exception you can extend Exception, however if you want to create unchecked exception you can extends RuntimeException

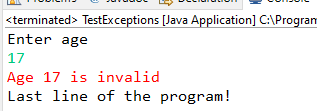
class Xyz extends Exception { } // checked exception

class Xyz extends RuntimeException { } // unchecked exception

Custom exceptions must be generated using throw keyword & you can handle it using try - catch block



Output:



File Handling

Java provides inbuilt API’s to work with file system, where you can read/write stream of data from simple to complex type of data you can main.

There are 2 types of streams

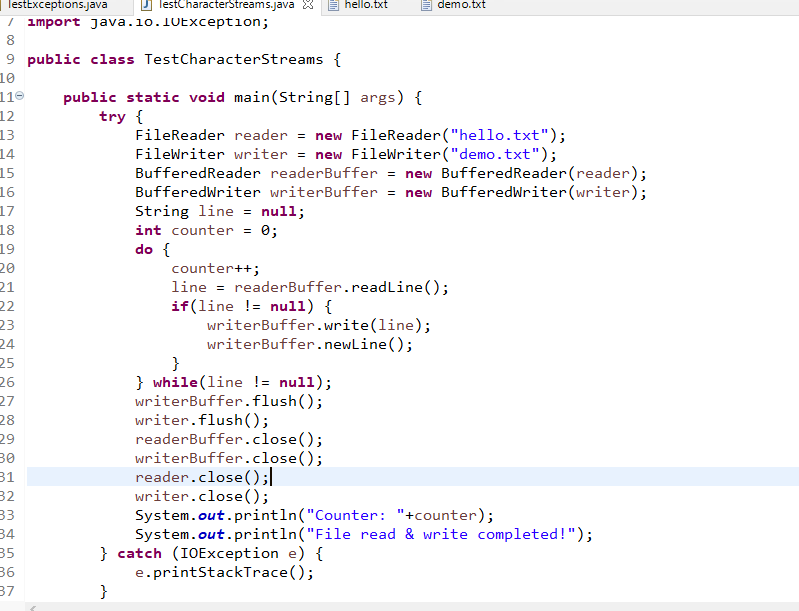
* 1. Byte streams : Read/Write binary data
  2. Character Streams: Read/Write character data/text data

We have set of API’s to work with Bytestreams

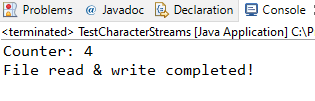
* 1. FileInputStream & FileOutputStream
  2. ObjectInputStream & ObjectOutputStream

We have set of API’s to work with Characterstreams

1. FileReader & FileWriter
2. BufferedReader & BufferedWriter



Output:



Serialization:

Serialization is nothing but writing the object to the file system, reading the object from the file system is known as deserialization.

By default not all objects can be serialized, you need to implement a marker interface called Serializable.

Serializable: It is a marker interface which will not have any methods, but adds a type to the object that implements, while writing the object the method would check if the object is of serializable type to write, if not it raises exception.

class User implements Serializable { }

Now User is of type Serializable & write method can successfully write the object to the file

class Employee { }

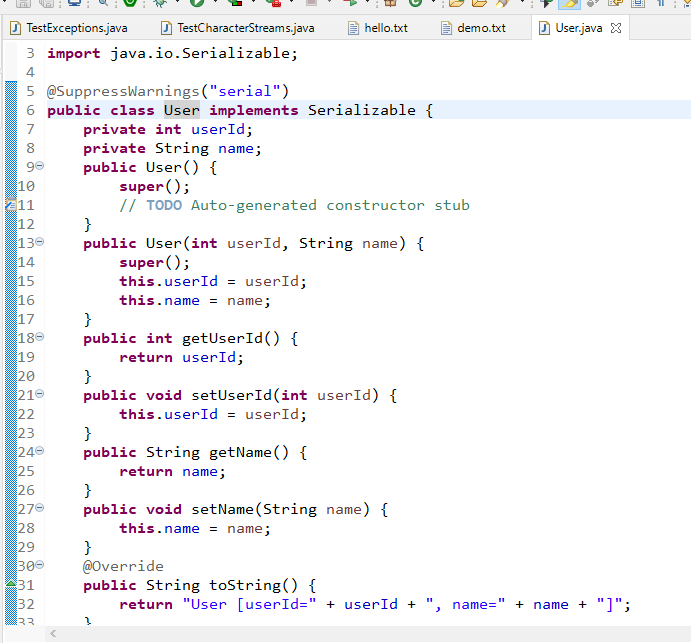
Now Employee class is not of type Serializable & write method can’t write Employee instance to the file.

We have 2 classes to perform serialization & deserialization

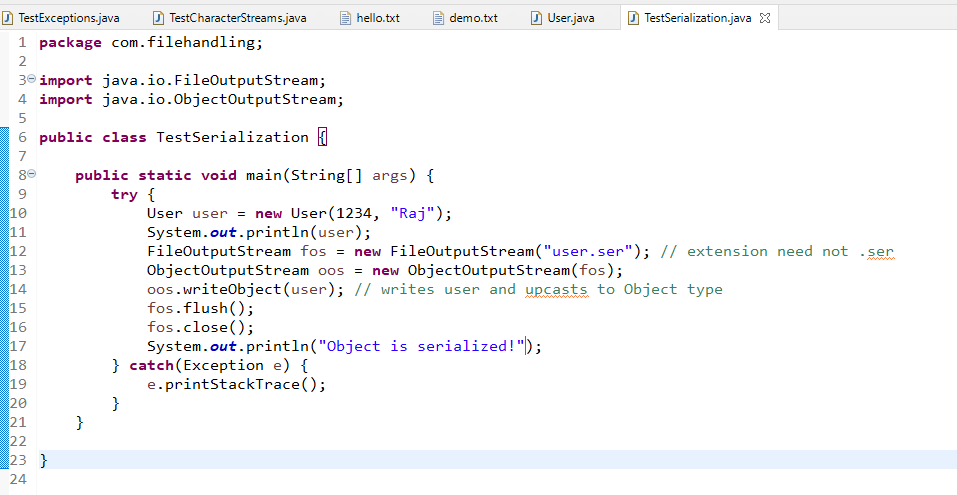
ObjectOutputStream: To write object, uses writeObject(Object) to write

ObjectInputStream: To read object, uses readObject() to read returns Object

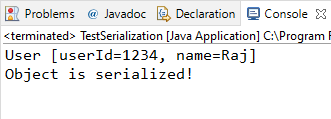
User.java



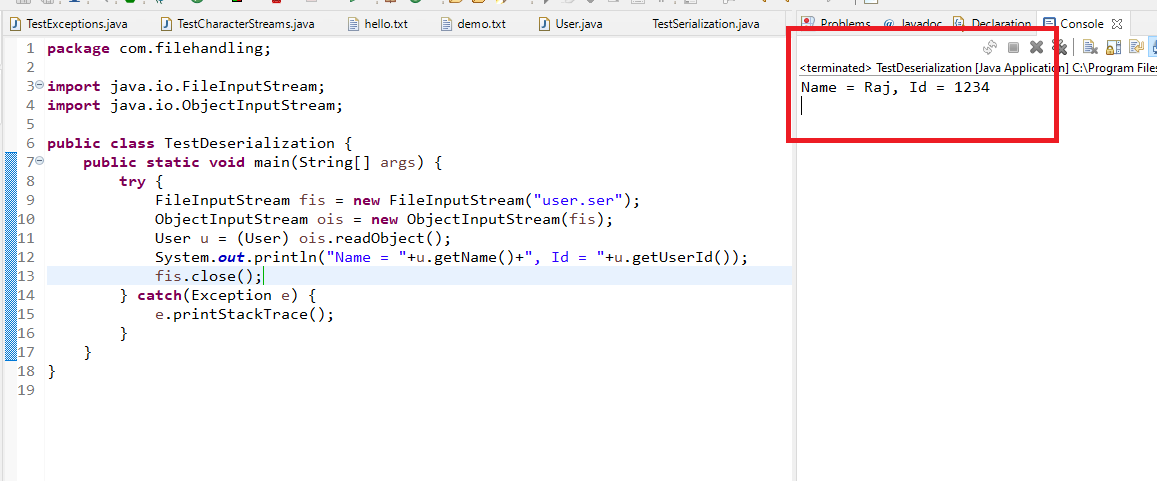
TestSerialization.java



Output:



TestDeserialization.java



Activity:

1. Demonstrate your understanding of the below concepts by writing necessary classes and methods

* Single try ... catch ... finally block
* Multiple catch blocks
* Nested try-catch blocks
* Usage of throw and throws keywords.

1. Create a checked exception UserNotFoundException. Override all the constructors from the super class
2. Create a User class with id, name and age properties, generate setters, getters, constructors, toString
3. Store some user objects in an array in a separate class from the main class and have a method called searchUser(int id) which accepts id as parameter, it must use throws to propagate UserNotFoundException if the passed id is not present in the array of users
4. From the main call searchUser(id) and pass the id, if id is present in the array of users, then it must print the user data, else it must throw exception that message must be printed in the catch block

Layered architecture

Whenever you develop any big applications, you need to separate Presentation, Navigation, Service and Database logics and use interface based approach to create a loosely coupled applications.

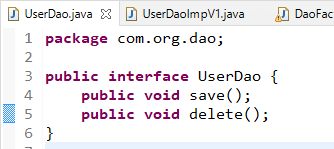
Loosely coupled: When you do changes in one place you don’t have to change the code in others machine, it can be achieved through some design patterns like factory pattern or singleton design pattern.

Factory pattern: It is a design pattern which takes care of creating the object by completely abstracting the object creation.

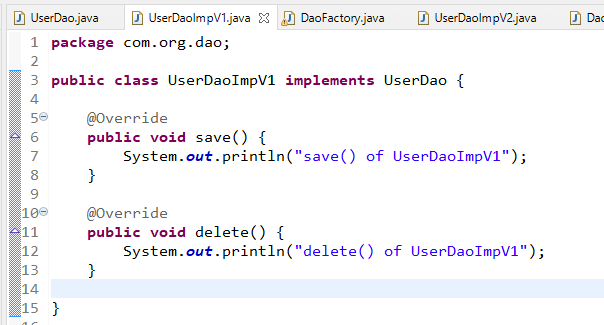
Note: It can return different implementations of same type based on some conditions

Singleton pattern: It is a type of factory pattern which creates only one instance of a class

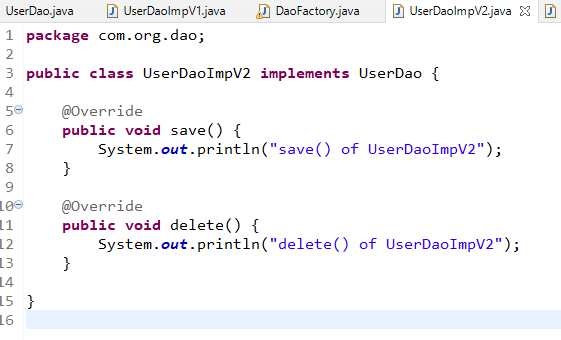
UserDao.java



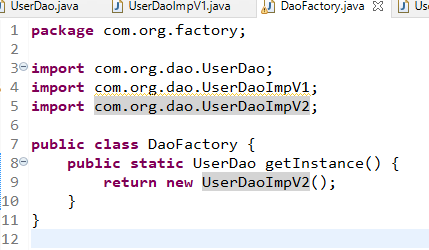
UserDaoImpV1.java



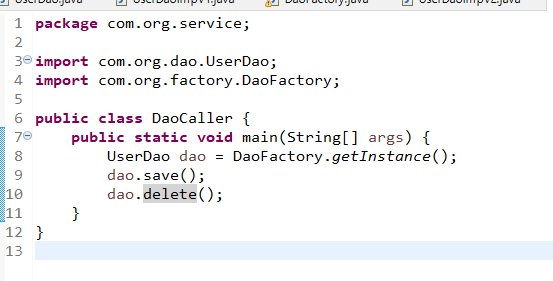
UserDaoImpV2.java



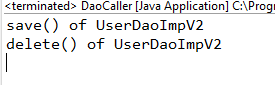
DaoFactory.java



DaoCaller.java



Output:



The above code is using interface & factory pattern to create a loosely coupled code, when you change the implementation the caller doesn’t need to change the code

Collection Framework

It provides set of API’s to order to maintain data (from simple to complex), the API’s maintains data in different way based on the requirement you can choose the one you need.

It is dynamic in nature whose size is not fixed and it uses Generics to maintain particular type of data.

Collection has an interface called Collection which provides methods to maintain data like add(), remove(), size(), isEmpty() and so on

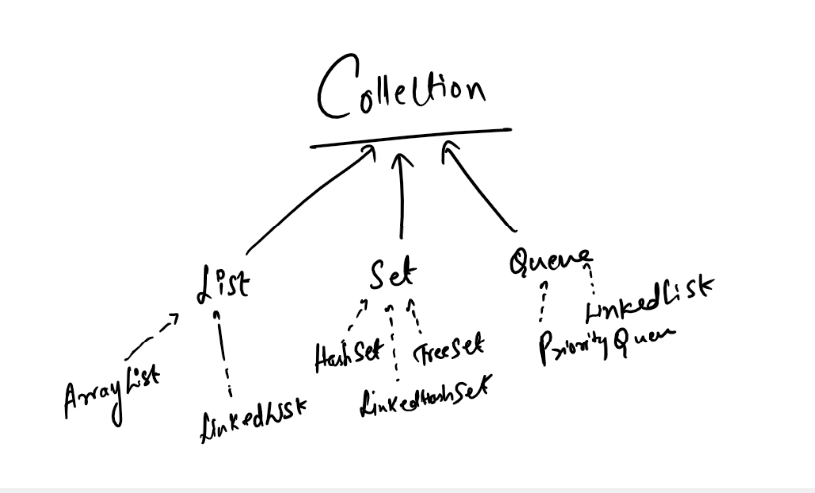
These methods are implemented by many classes like LinkedList, ArrayList, HashSet, LinkedHashSet, TreeSet, PriorityQueue and so on.

Since when you want to maintain data you may sometimes prefer unique data no duplicates, data in sequential order, random order, sorted order, First In First Out, data supporting duplicates, data supports index based access the Collection itself divided into 3 interfaces which are sub-interface of Collection

* 1. List: Supports duplicates & index based access
  2. Set: Supports only unique elements
  3. Queue: Supports First In First Out & Last in Last Out based access

All these 3 interfaces are implemented by specific classes which are

1. List
   1. ArrayList
   2. LinkedList
2. Set
   1. HashSet
   2. LinkedHashSet
   3. TreeSet
3. Queue
   1. PriorityQueue
   2. LinkedList (It implements List & Queue)



ArrayList: Stores elements in contiguous memory address, here retrieval will be faster whereas add and remove is slower

LinkedList: Stores elements in non-contiguous memory address, here add & remove is faster & retrieval will be slower

Both of them supports duplicates

LinkedList also implements Queue, hence it supports methods to perform First in First out, the methods are add() to store item, poll() is to remove item in first in first out

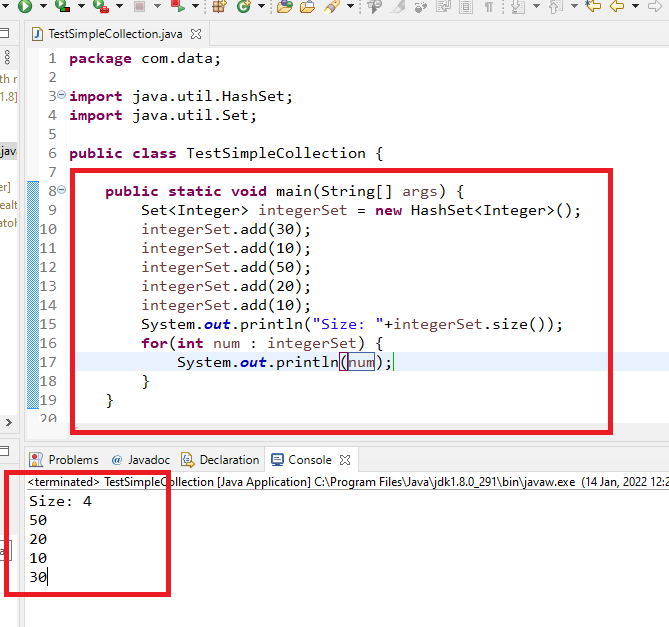
PrirorityQueue: It removes the elements in sorted order, based on priority it removes the elements, it also supports duplicates

Set: It only supports unique elements

HashSet: It maintains elements in random order whereas the elements can be retrieved faster as it uses hash algorithm i.e., based on a particular key it can search the item.

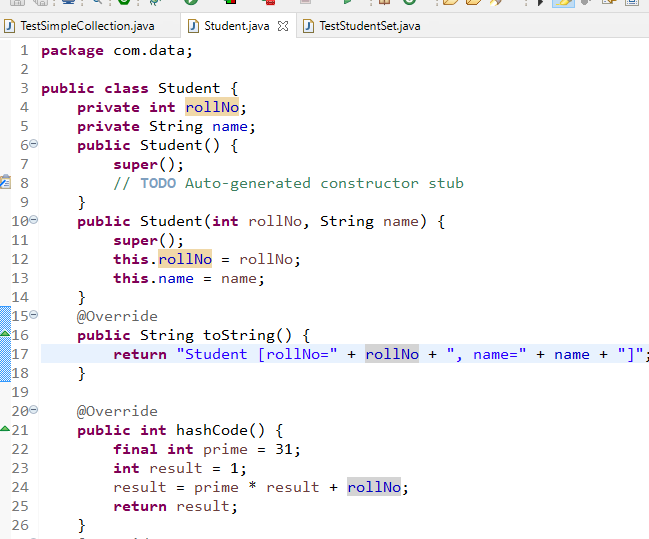
LinkedHashSet: It maintains elements in insertion order, It is like First Come First Serve

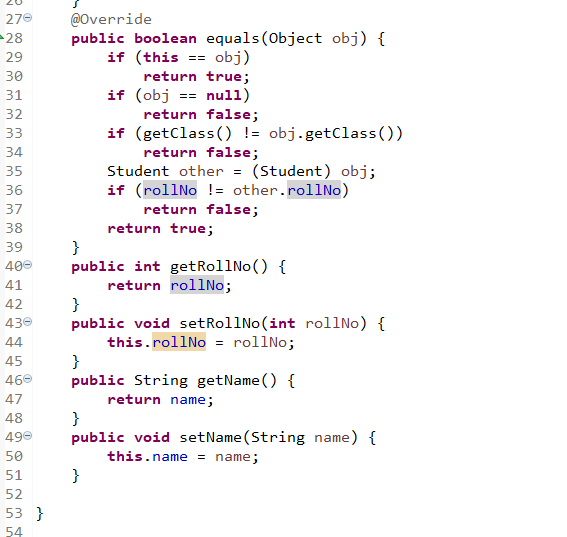
TreeSet: It maintains elements in sorted order



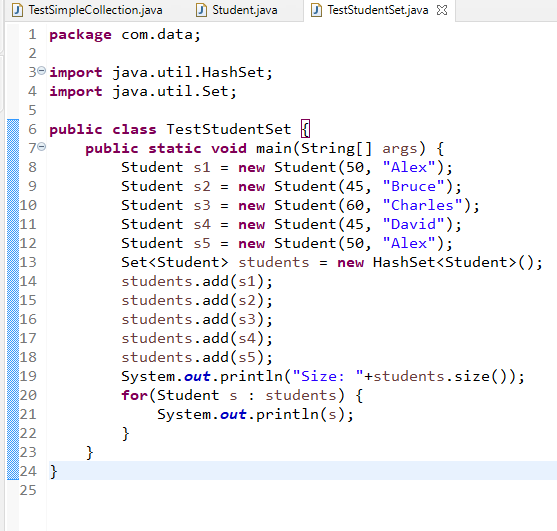
You usually maintain complex types like Customer, User, Employee, Product & so on instead of simple types like int, String

Student.java

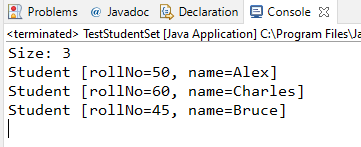




TestStudentSet.java



Output:

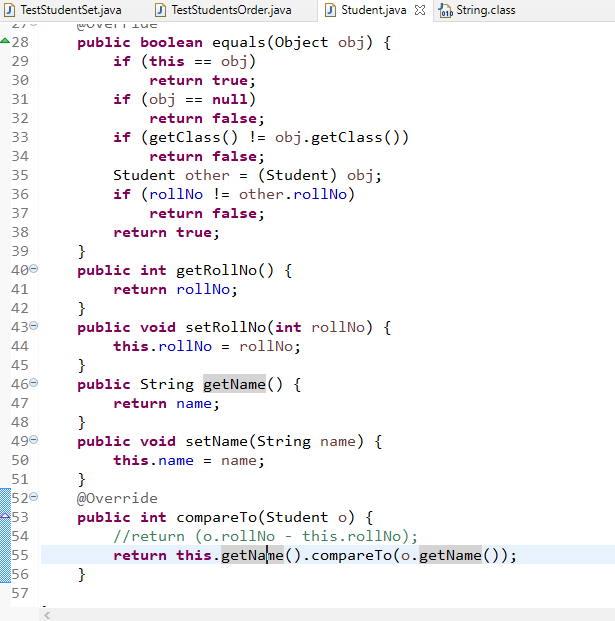
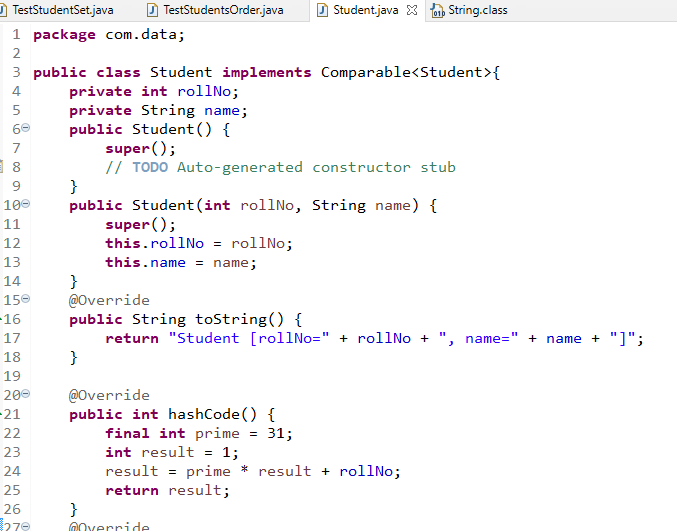


TreeSet:

It maintains the data in sorted order, it can sort the simple types automatically, but if its complex types it needs the object to either implement Comparable or Comparator.

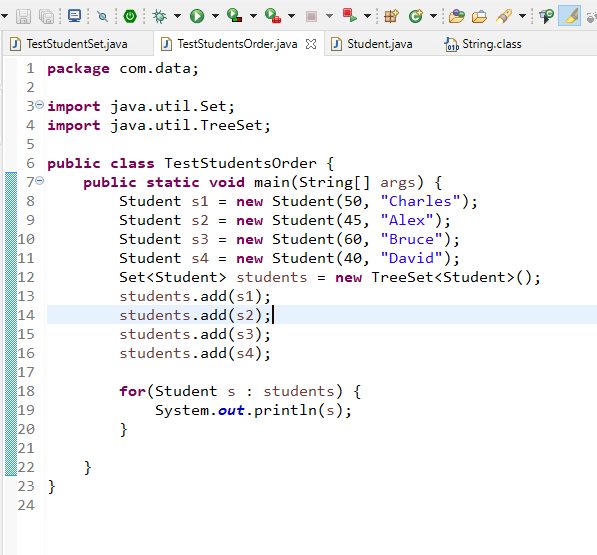
Comparable: It is an interface which has 1 method compareTo() which is called when two objects need to be compared, it returns an int value, if negative then object is treated as lower than the compared one, if 0 object is treated as same if positive then object is treated as higher than the compared one, TreeSet needs this int value when 2 objects are compared, based on that it can sort.

Student.java

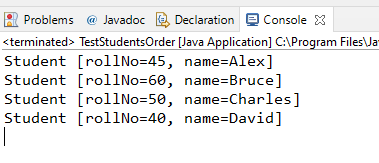


Output:

TestStudentsOrder.java



Output:



Since Student comapreTo is comparing name, the students are sorted based on name.

Suppose you have other types then you need to find a way to return an int,

ex: If two values are float, you have a static compare method in Float class, Float.compare(f1, f2), it returns -1, 0, +1 based on 2 parameters

ex: if two values are string, you can use compareTo method of string, s1.compareTo(s2)

ex: If two values are double, then use Double.compare(d1, d2), here 2 arguments must be double & returns int

ex: if two values are int, you can use Integer.compare(i1, i2), here 2 arguments must be int & returns int, or you can use i1 - i2

But in Student class you can only provide order for any one properties not for all the properties, if you want to sort rollNo & name both in ascending & descending order you can’t use Comparable, for that reason you need to use Comparator.

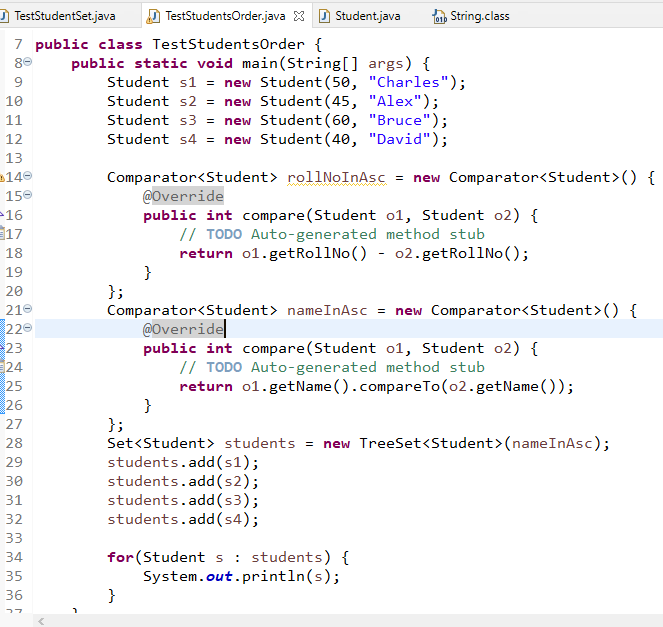
Comparator: It is an interface which has compare() method takes 2 arguments which needs to compared, it can be written independently from the instance that needs to be compared

i.e., Comparable needs to be implemented by the same class whose instance needs to be compared.

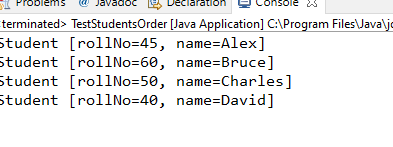
however Comparator can be implemented in a separate class & use that comparator instance in TreeSet or you can also use in Collections.sort(), it is used to sort List items.

Comparable vs Comparator

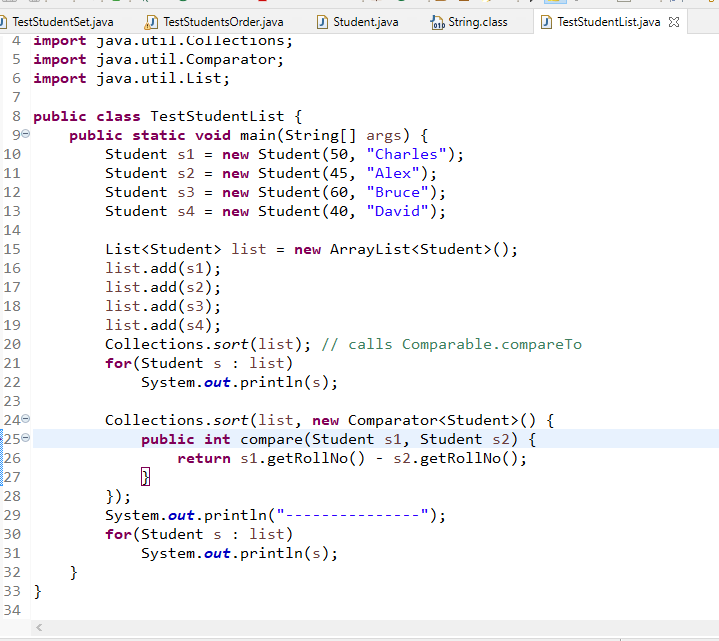
|  |  |
| --- | --- |
| Comparable | Comparator |
| It must be implemented in the same class whose instance needs to be created | It can be implemented in different class |
| compareTo() | compare() |
| Natural ordering - default comparison | Multiple sorting technique based on the number of implementations you created |



Output:



You can also use sorting techniques for List items



Activity:

* 1. Create an Employee class with following properties
     + id: int
     + name: String
     + salary: double
  2. Create necessary getters & setters, including toString()
  3. Create an EmployeeStorage interface which will have following methods
     + public int save(Employee)
     + public Employee findEmployee(int id) throws EmployeeNotFoundException
     + public Employee[] findEmployees()
     + public void delete(int id)throws EmployeeNotFoundException
  4. EmployeeNotFoundException must be a checked exception & must be generated if the id is not present, that must be propagated to the UI layer and print if the id is not present
  5. Implement the EmployeeStorage interface in a class called EmployeeStorageImpl, it uses ArrayList to maintain the employee object.
     + save(Employee): stores Employee object & return id of the employee
     + findEmployee(int): Accepts id if its present in the ArrayList return the employee matching the id else throw EmployeeNotFoundException
     + findEmployees(): returns all the employees in the ArrayList in array format
     + delete(int): deletes the employee present in the ArrayList if id is found else throws EmployeeNotFoundException
  6. Create a main method in a separate class that uses EmployeeStorage as the reference & gets the object of EmployeeStorage implementation using factory pattern, the main methods shows below options repeatedly until you enter option: 5
     1. Store
     2. Find by id
     3. Find all
     4. Delete by id
     5. Exit

option 1: Must ask user to enter id, name & salary and call save() method from EmployeeStorage, the generated id the save() returns must be printed in main method

option 2: Ask id from the user & call findEmployee(id), if it returns Employee print that employee information, if method generates exception print that exception message

option3: call findAll() & iterate the array and print all the employee informations

option4: Ask id from the user & call delete(id), if id is not present print the exception message, else print the successful deletion message

option5: exits from the application.

Map: It is used to maintain the data in key value pairs, Map is an interface with methods like put(key, value), get(key), remove(key). It is similar to the Set api but Map maintains key & value both, whereas Set uses only value

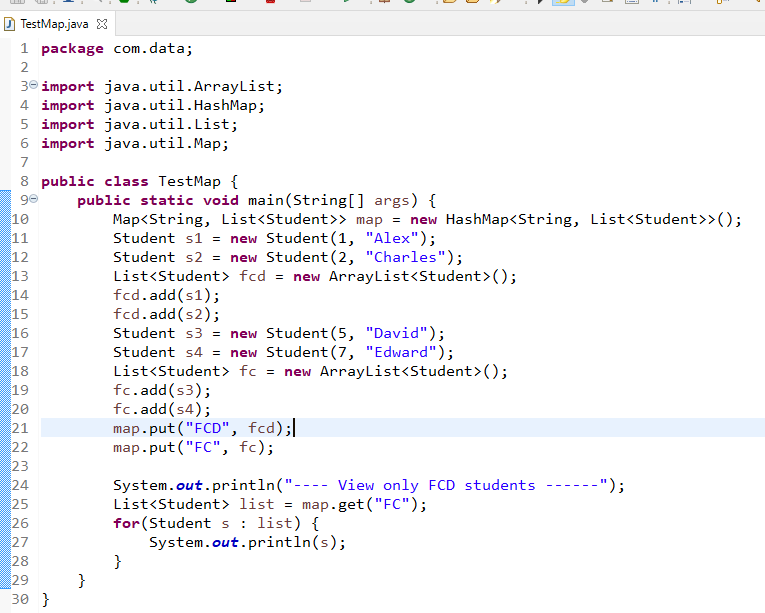
It is implemented by following classes like

* 1. HashMap: Maintains the data in random order, but it is newer than Hashtable, it supports null
  2. Hashtable: Maintains the data in random order, but it is older class & doesn’t support null
  3. LinkedHashMap: Maintains the data in insertion order
  4. TreeMap: Maintains the data in sorted order

put(key, value): It is used to store key & value

get(key): It is used to retrieve the value based on the key

remove(key): It is used to remove the value



JDBC

JDBC provides set of API’s to interact with the database from the Java program, JDBC works with any Relational Database (RDBMS), every RDBMS has implemented JDBC API’s in their driver so that you can use those drivers to interact with their database.

Some of the interfaces from JDBC API are:

Connection

Statement

PreparedStatement

ResultSet

As a Java Developer you need to only use these interfaces & call their methods, as you are using the Drivers of appropriate databases, these interfaces call the methods implemented by these drviers

Every database provides JDBC implementations in the form drivers as a jar file, you need to download from the internet, either manually or through maven project.

But everyone prefers to use Maven hence we will also use Maven.

Steps to interact with any database

1. Loading the driver

Class.forName(className)

1. Establishing the connection:

Connection con = DriverManager.getConnection(url, username, password)

1. Statement: takes Java datatypes and converts to SQL datatypes

Statement stmt = con.createStatement()

1. Executing the statements

int x = stmt.executeUpdate(query) // DML (insert, update, delete)

ResultSet rs = stmt.executeQuery(query) // DRL (select)

1. Closing the resources

rs.close();

stmt.close()

con.close()

Connection: It is an API which allows you to establish connection via DriverManager.getConnection(), the getConnection() takes 3 arguments

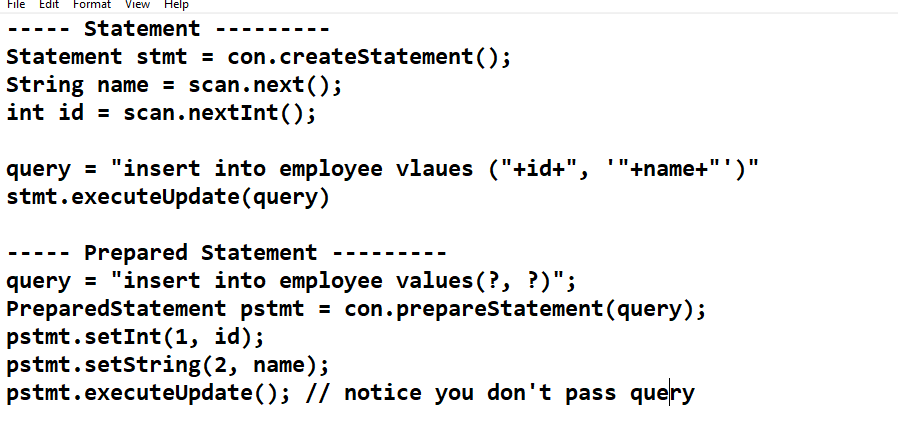
url: You must provide database URL

username: You must provide database username

password: You must provide database password

Statement: It is an object that provides methods to execute SQL statements & also takes care of converting Java types to long types, In JDBC there are 3 types of interfaces that provides methods to execute SQL Statements

1. Statement: It is used to execute queries which are not dynamic.
2. PreparedStatement: It is used to execute queries which are dynamic
3. CallableStatement: It is used to invoke stored procedures(programs that are stored in the database)



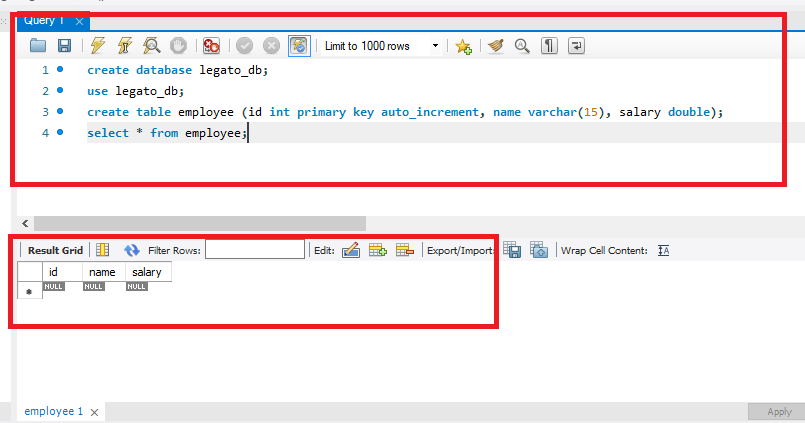
ResultSet: It can navigate over the results produced from the select query, it has many methods like next(), getter methods to get the value from the ResultSet i.e.,

getString(columnIndex): returns the varachar in String

getInt(columnIndex): returns the number in int

getFloat(columnIndex): returns the number in Float

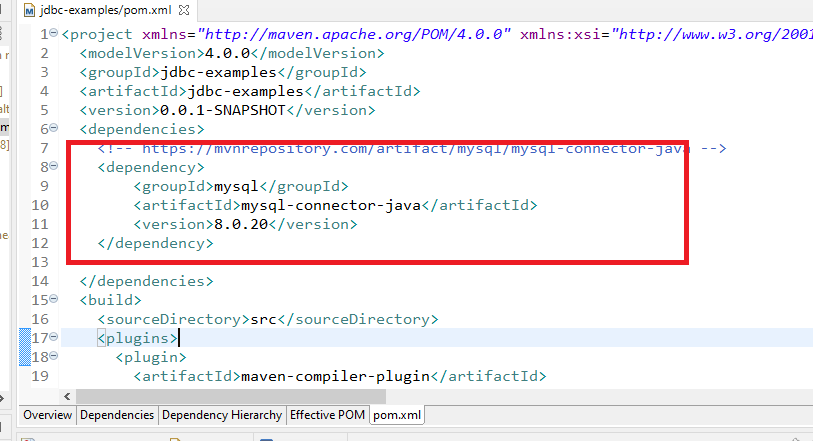
Create a database instance & a table called employee in MYSQL



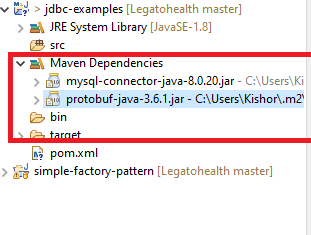
Create Java Project & Convert to Maven Project

Add mysql-connector dependency in pom.xml

pom.xml



You could see Maven Dependencies

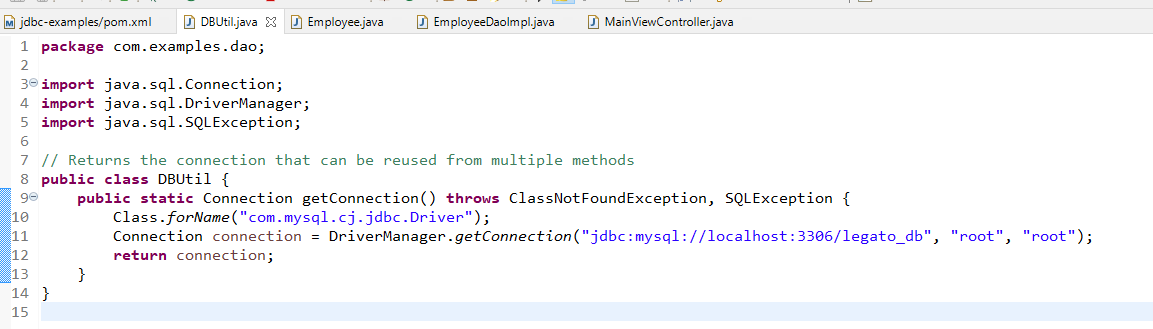


Since inorder to interact with the database to perform any CRUD operations you need to mandatorily perform 2 steps,

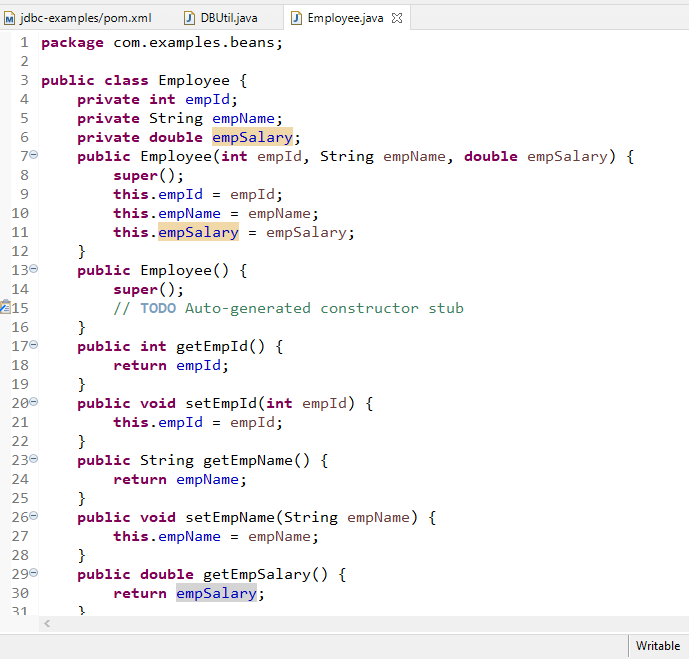
* loading the driver
* establishing the connection

We can reuse the above 2 steps by creating DBUtil.java class that returns connection object

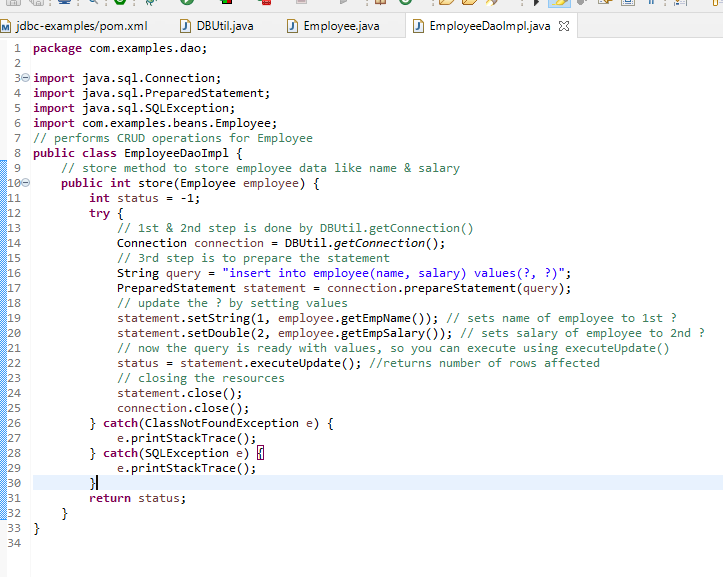
DBUtil.java



Creating a class that represents employee table



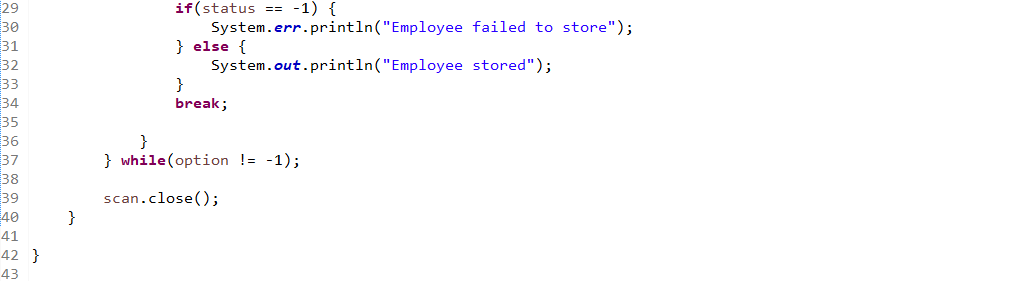
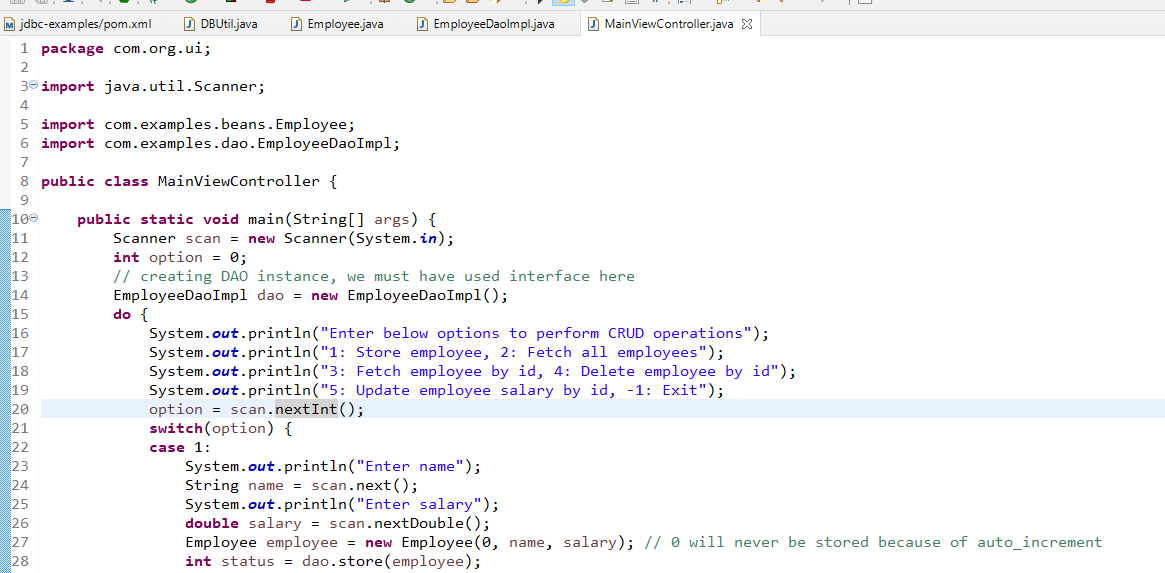
Creating a DAO class to perform CRUD operations



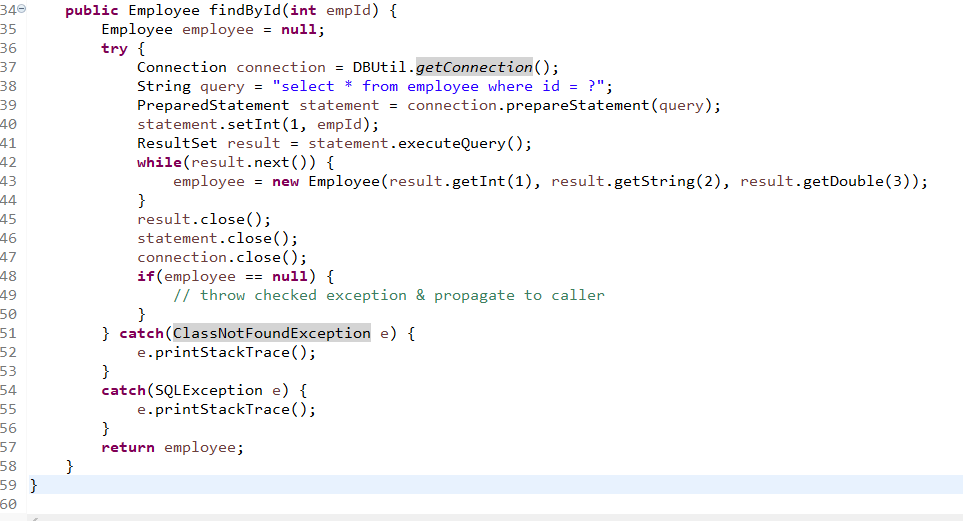
Note: Above code needs to use interface and implement those methods, but to make the code simple, we haven’t used the interface & also not used finally block to close the resources.

Calling the store from the caller (i.e, main as of now, but it must be a service layer)

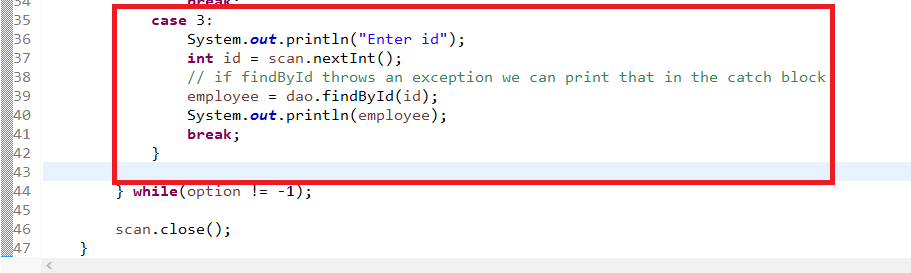
MainViewController.java



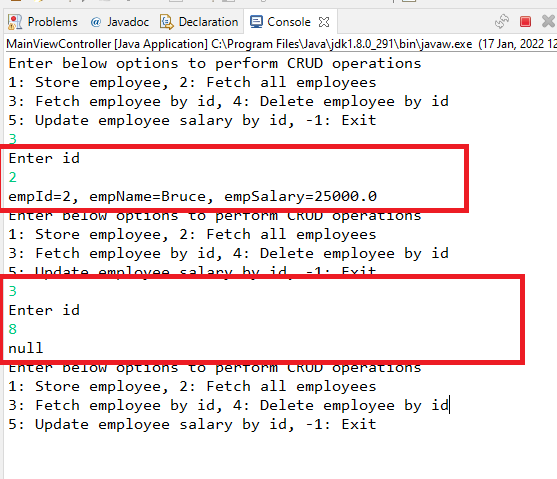
Now you can create another method findById in EmployeeDaoImpl and return the employee matching to the id



Call this method from MainViewController.java

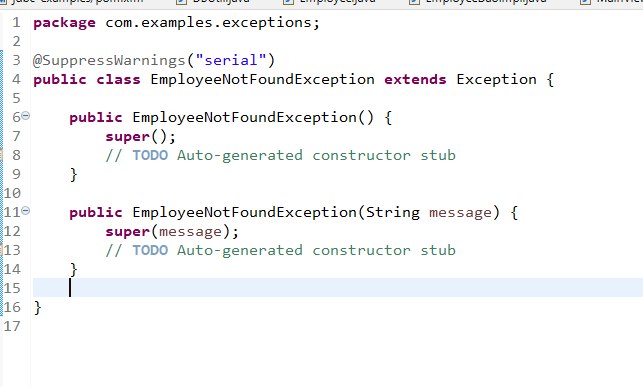


Output:



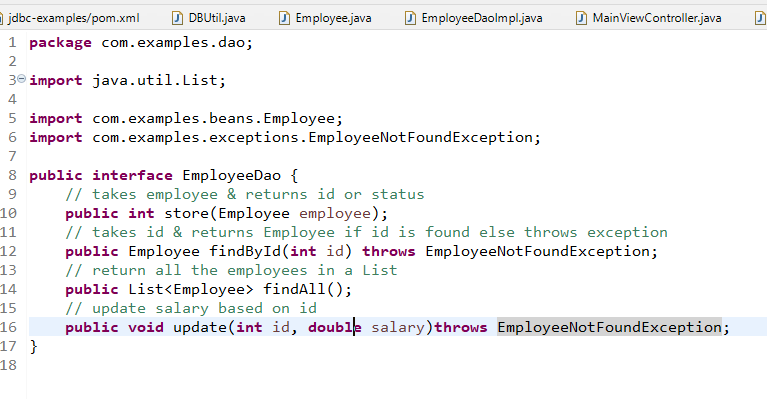
Creating a custom exception called EmployeeNotFoundException

EmployeeNotFoundException.java

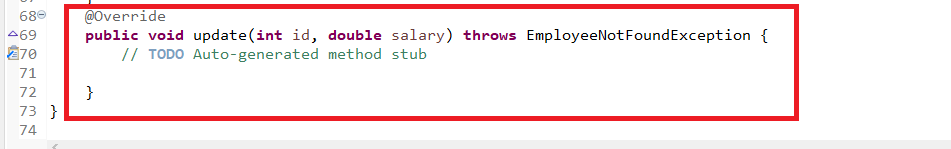
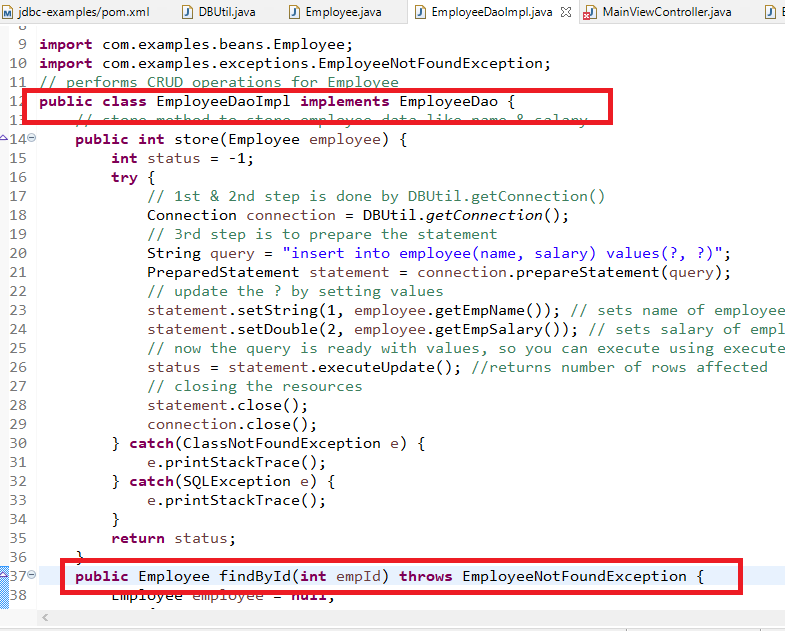


Above implementation needs to use interface hence we will create an interface & use the interface methods instead of directly instantiating the class.

EmployeeDao.java

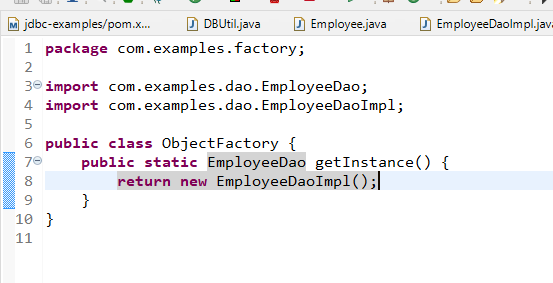
Modify the EmployeeDaoImpl to implement this interface

EmployeeDaoImpl.java

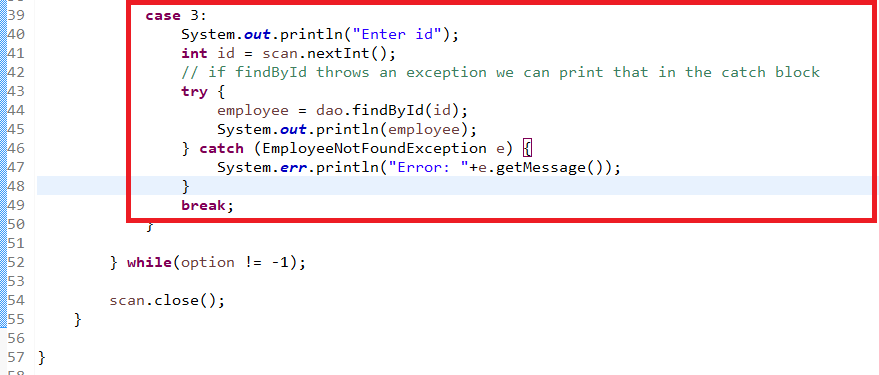
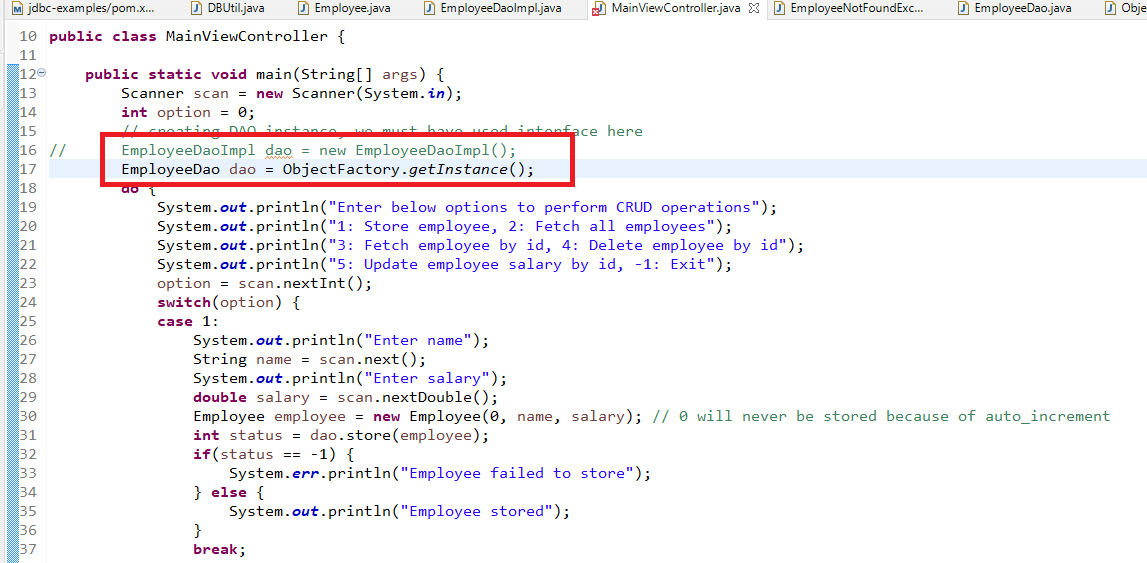


Since the interface is used now, we also need to use interface reference in the main method & also instantiate this implementation via factory pattern.

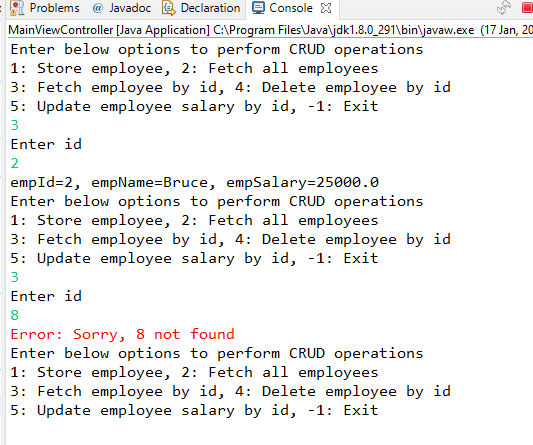
ObjectFactory.java



Lastly the ViewController can use this factory pattern & EmployeeDao interface reference & handle exception if required.



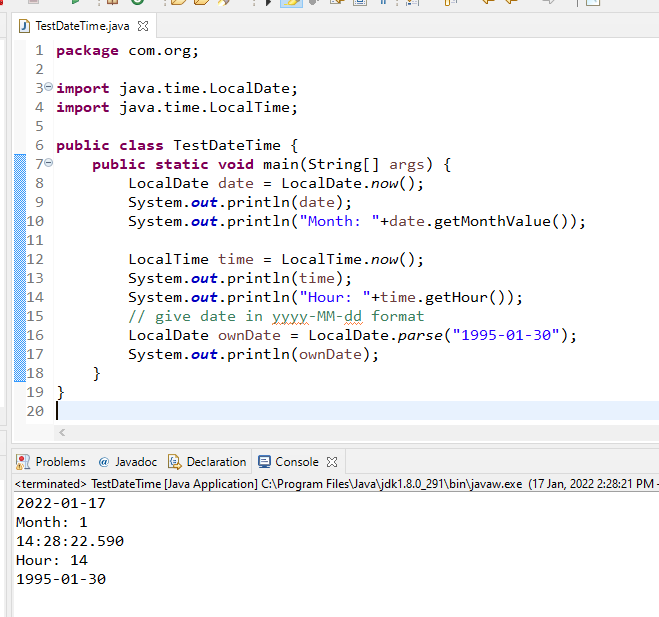
Output:



Java 8 features

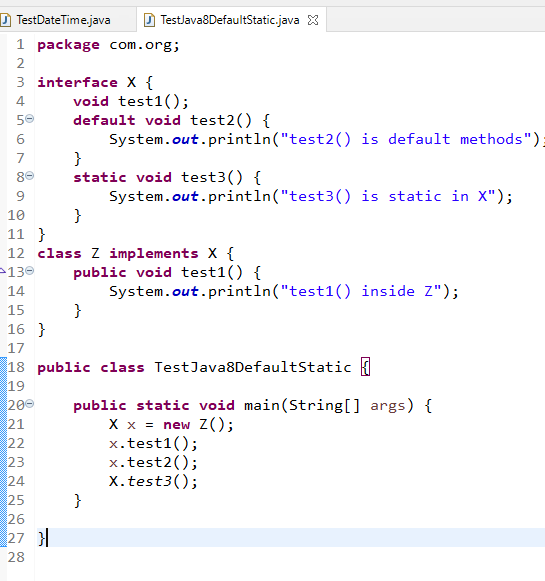
* Static & Default methods in interface
* Functional interface & Lambda expressions
* New Date & Time api’s (LocalDate, LocalTime, LocalDateTime)
* Stream API

LocalDate & LocalTime: They give the correct result unlike old Date, Calendar classes which were giving wrong results ex: for January they give 0, instead of 1, getYear() returns 122, if its 2022, but the new date & time classes gives the correct result.



Static & Default methods in the interface

This is to write extra methods in an already existing interface without breaking the contract, i.e., all the classes implementing the existing interfaces need not be changed



Functional interface

These are the interface which will have only one abstract method to support passing function as an argument so that it enables functional programming.

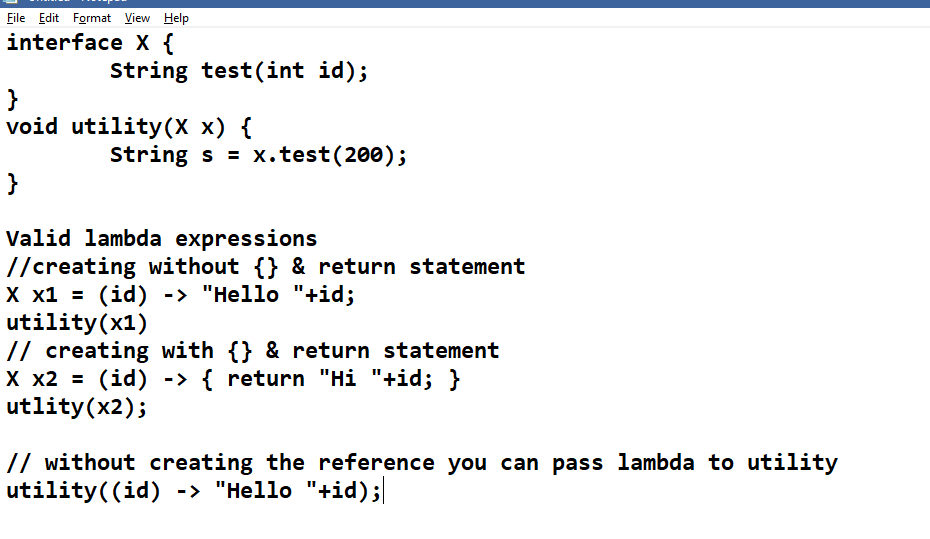
Till now we passed object, but from Java 8 onwards you can also pass function as an argument, but it is possible only if the method can accept function interface reference.

With Lambda expression you can implement functional interface, so that it doesn’t need to specify it has to implement which method, as it is understood that the interface has only one abstract method & what is its signature, the lambda expression doesn’t need to explicitly specify that.

ex:

interface X { void test();}

X x = new X() {   
 public void test()   
}



In Java there are lot of functional interfaces like

Comparator

Runnable

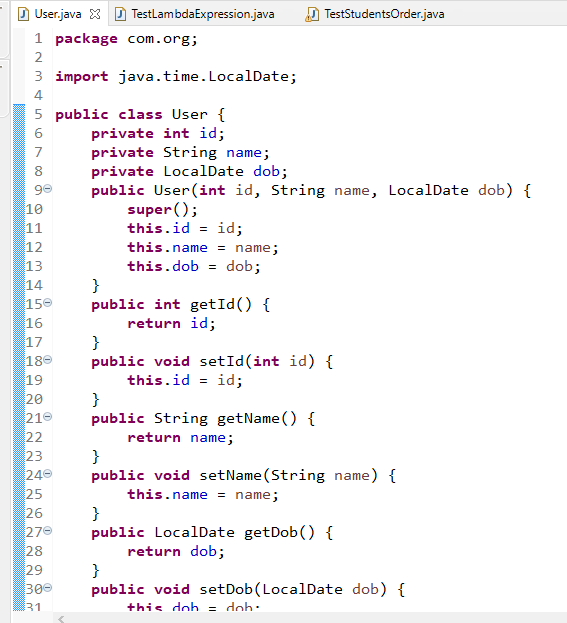
Functional interface introduced in Java 8

Predicate

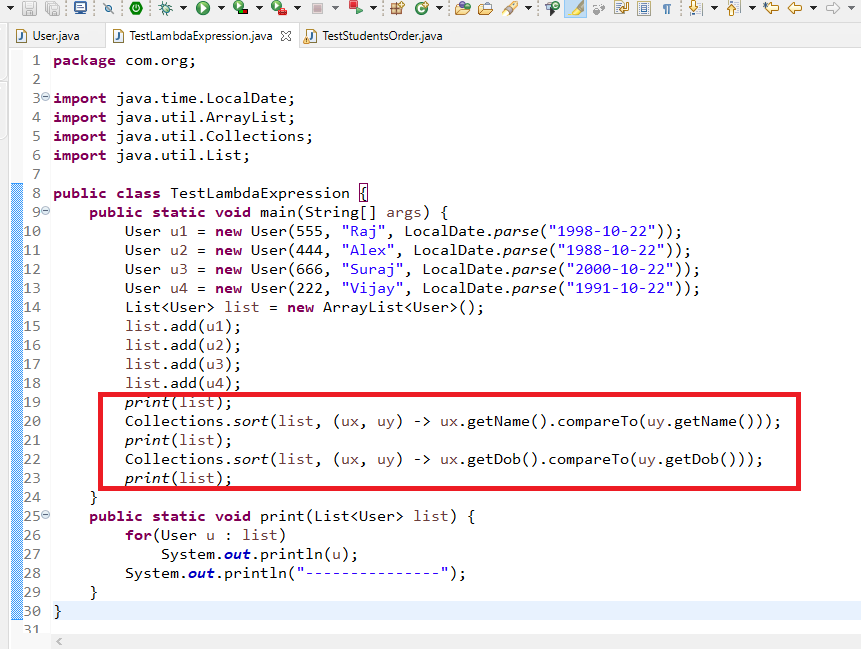
Consumer

Supplier

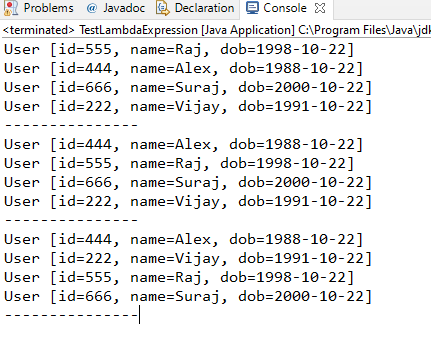
User.java



TestLambdaExpression.java



Output:



Activity:

Make use the existing program written for EmployeeDao and implement other two methods and call them from the main method.

Java 8 Streams

It is used to process the bulk data in a declarative way without using too much code on iteration or conditions

Declarative way is more easier compare to writing logics, ex: the SQL statements are easier to write & understand because of its declarative syntax.

Similarly Java 8 allows you to process collection of data in a declarative way, through lambda expressions.

Java 8 Stream allows you to limit iterations and writing conditional statements, itself provides methods that takes functional interface as parameter, so you can pass functions or lambda expressions to it.

When you want to process the data & show at the user end you will come up with lot of logics related to filtering, iterating, sorting, mapping, counting and so on.

In Collection Framework you can do that suppose, you have some laptops & you want the laptops on below conditions so that each conditions derive a separate list so that you can work on that list if required, hence you don’t modify the existing list or collection.

Suppose below is the requirement

1. Laptops ram size > 8
2. Using above filtered laptops you need Laptops except HP
3. Using the above filtered laptops you want price to be sorted

Then below is the code you need to write



Above code is using too must iterations and creating new lists so that original lists wouldn’t be affected.

Same requirement you can achieve in streams with a lesser number of codes, because of declarative approach using lambda expression.

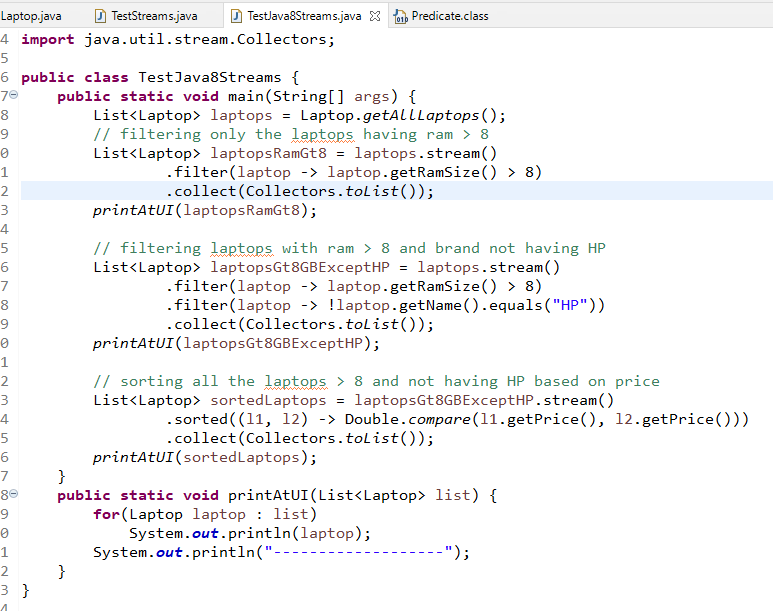
Stream provides many methods to perform different operations like

filter(), sort(), collect(), forEach(), map(), count() and so on

All the above methods are present in streams, some of them returns new streams and some are terminal operations which doesn’t return streams.

How to apply streams

stream(): present in every collection, it creates a stream of data & any operation you perform doesn’t affect the existing collection.



Above code does the same job without any for loop or if conditions, because stream methods does internal iterations, and also they are faster.

We have two types of operations in streams

1. Intermediate operations: They return another stream, it can again call another intermediate operation or terminal operation

ex: filter(), sorted(), map()

1. Terminal operations: They are the end of the stream, you can’t call any further operations

ex: collect(), count(), forEach()

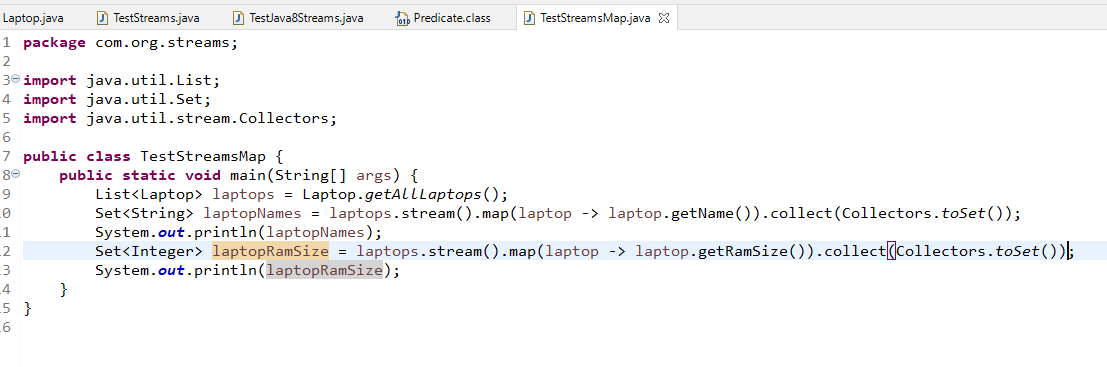
Filter: It creates a new stream by filtering the incoming stream with some conditions

Sort: It can sort the data coming from the stream and creates a new stream having sorted data

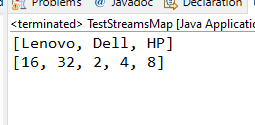
Map: It can take a stream of one type and produce a stream of different types

ex: You want to have a separate list having only the name of the laptops from the Collection having laptops

laptops.stream().map(laptop -> laptop.getName()) // this takes laptop object returns a new stream having only names



Output:



Activity:

1. Try above stream examples
2. Use count() method of streams to show the size of data in a stream
3. Use forEach() method of stream to iterate and print in the console
4. Like stream() there’s a parallelStream() use that and observe the output you get